Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Conception et réalisation des applications de gestion : une approche basée sur l'explicitation des raisonnements

Mathieu, Isabelle

Award date:
1986

Awarding institution:
Universite de Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 04. Jul. 2025

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX - NAMUR

INSTITUT D'INFORMATIQUE

CONCEPTION ET REALISATION DES APPLICATIONS DE GESTION : UNE APPROCHE BASEE SUR L'EXPLICITATION DES RAISONNEMENTS

Isabelle MATHIEU

Mémoire présenté en vue de l'obtention du grade de licenciée et maître en informatique

Année académique 1985 - 1986

PLAN

page

INTRODUCTION

PREMIERE PARTIE : DEUX PROPOSITIONS D'AIDE AU DEVELOPPEMENT D'APPLICATIONS DE GESTION

Introduction 1

Chapitre 1 : Explicitation des raisonnements 2

1. Présentation générale 2

1.1. Origine : Les méthodes de démonstration de programmes 2

1.2. Généralisation des idées sous-jacentes aux méthodes de démonstration de programmes : explicitation de tous les raisonnements. 3

2. Spécification 5

2.1. Introduction 5

2.2. Sens généralement accordé au concept de spécification 5

2.3. Pourquoi l'analyste se contente-t-il de fournir les "résultats" ? 6

2.4. Pourquoi cette conception de la spécification n'est-elle pas satisfaisante ? 7

2.5. Proposition d'une démarche de spécification. 7

2.5.1. Introduction 7

2.5.2. Spécification de l'application ou position du problème 8

2.5.2.1. En quels termes l'analyste doit-il poser le problème ? 8

2.5.2.2. Sur quoi porte le travail de l'analyste et où s'arrête-t-il ? 9

2.5.2.3. Enumération d'un certain nombre de tâches ressortant du domaine de l'analyse 9

2.5.3. Spécification des programmes 11

2.5.3.1. Décomposition d'un problème en sous-problèmes et critère de décomposition 11

2.5.3.2. Qu'est-ce qu'un énoncé simple ? 12

2.5.3.3. Les spécifications de programmes sont des énoncés simples 12

2.5.3.4. Que doit contenir une spécification de programme ? 13

page

2.5.4. Remarque sur les deux sens donnés au mot spécification. 14

3. Construction 16

3.1. Présentation de la méthode utilisée dans la deuxième partie. 16

3.1.1. Forme des spécifications 16

3.1.2. Forme des programmes 17

3.1.3. Principes de la méthode 18

3.2. Illustration 19

3.2.1. Enoncé 19

3.2.2. Exemple 20

3.2.3. Notations 20

3.2.4. Spécification sous forme de situation initiale, situation finale 21

3.2.5. Construction 21

3.3. Applicabilité pratique de la méthode proposée 27

4. Tests 29

4.1. Raison d'être des tests 29

4.2. Principe de construction des tests 30

4.3. Illustration 30

4.3.1. Cas des suites à un élément 31

4.3.2. Cas des suites à deux éléments 31

4.4. Avantage de la méthode proposée 32

4.5. Détermination des hypothèses 32

4.5.1. Décomposition en sous-problèmes 32

4.5.2. Formes des programmes 33

Chapitre 2 : Seconde proposition : approche orientée modèles-outils-règles 34

1. Introduction 34

page

2. Une méthode de spécification des applications informatiques de gestion : l'analyse fonctionnelle 34

2.1. Introduction 34

2.2. La notion de schéma conceptuel 35

2.3. Utilité de l'analyse fonctionnelle-difficultés 36

2.4. Les modèles et les outils 36

2.4.1. Le modèle entité-association(description des informations) 36

2.4.1.1. Remarques préliminaires 36

2.4.1.2. Contenu conceptuel de la base des données : entités, associations et valeurs 37

2.4.2.3. Structuration de la base de données : types d'entités, types d'associations et propriétés 37

2.4.2.4. Contraintes d'intégrité 39

2.4.2.5. Définition des contenus conceptuels possibles de la base de donnée : le schéma Entité-Association 39

2.4.2.6. Description graphique 40

2.4.2. Les modèles des traitements 41

2.4.2.1. Le modèle de structuration des traitements 41

2.4.2.2. Le modèle de la dynamique des traitements 42

2.4.2.3. Le modèle de la statique des traitements 43

2.4.3. Le modèle des ressources 43

2.4.4. Les outils automatisés 43

2.5. Commentaires 44

2.5.1. Outil de spécification 45

2.5.2. Outil de validation 45

3. Méthode de construction de programmes : la programmation structurée de Jackson (P.S.J.) 46

3.1. Présentation précise 46

3.1.1. Types de programme considérés 46

3.1.2. Représentation des programmes 46

page

3.1.3. Structuration des données 48

3.1.3.1. Types de composants 49

3.1.3.2. Illustration des difficultés posées par la structuration des données 51

3.1.4. Définition de la notion de correspondance 54

3.1.5. Construction du programme 55

3.1.6. Détermination des conditions et actions primitives 56

3.1.7. Illustration 59

3.1.7.1. Structuration des sorties 60

3.1.7.2. Recherche d'une correspondance entre A1 et B 61

3.1.7.3. Recherche d'une correspondance entre A2 et B 64

3.2. Limites et extensions possibles 69

3.2.1. Cas de plusieurs flux d'entrée et de sortie 69

3.2.2. Les conflits de structure 69

3.2.3. Solutions aux conflits de structure 70

3.2.4. Inversion de programme 72

3.2.4.1. Principes 72

3.2.4.2. Règles d'inversion et justification 72

4. Tests 78

4.1. Tests construits à partir des spécifications 78

4.2. Tests construits à partir de l'algorithme 79

4.3. Tests d'intégration 79

page

DEUXIEME PARTIE : APPLICATION 80

Introduction 80

Chapitre 1 : Spécification 81

1. Description de la procédure DOM80 81
1.1. Présentation générale 81
1.2. Description sommaire de la procédure DOM80 81
1.3. Faits et définitions 82

2. Problème du contrôle 84

3. Implémentation de la solution au problème du contrôle 88
3.1. Partie de l'existant à modifier 88
3.2. Impact de la solution au problème du contrôle sur cette partie 90
3.3. Organisation physique des informations nécessaires au contrôle 93
3.4. Représentation des entrées 94
3.5. Représentation des sorties 97
3.6. Spécification du programme contrôle 98

Chapitre 2 : Construction du programme contrôle 99

1. Construction selon la méthode de Jackson 99
1.1. Spécification du programme contrôle 99
1.2. Structuration des entrées et sorties du programme contrôle 99
1.3. Recherche des correspondances - conflit de structure et solution 102
1.4. Spécification de P1 102
1.5. Structuration des entrées et sorties de P1 103
1.6. Recherche des correspondances 105
1.7. Ajout des conditions et actions primitives 106
1.7.1. Spécifications des différents composants 106
1.7.2. Détermination des conditions et actions primitives 109
1.7.2.1. Détermination des conditions d'itération 109

page

1.7.2.2. Détermination des conditions de sélection 111

1.7.2.3. Détermination des actions primitives 112

1.8. Spécification de P2 113

1.9. Structuration des entrées et sorties de P2 113

1.10. Recherche des correspondances 115

1.11. Ajout des conditions et actions primitives 116

1.11.1. Spécifications des différents composants 116

1.11.2. Détermination des conditions et actions primitives 118

1.11.2.1. Détermination des conditions d'itération 118

1.11.2.2. Détermination des conditions de sélection 119

1.11.2.3. Détermination des actions primitives 119

1.12. Inversion 119

2. Construction selon l'approche "Explicitation des raisonnements" 120

2.1. Remarques 120

2.2. Descriptions des zones Cobol 120

2.3. Définitions 120

2.4. Spécifications des sections 126

2.5. Construction et justification de 2 sections 129

2.5.1. Section P-GROUPE-CEC 129

2.5.2. Section P-GROUPE-CREANCIER-CEC 132

Chapitre 3 : Test 139

1. Construction du premier ensemble de jeux de données de test 139

1.1. Premiers jeux de données de test 139

1.2. Construction de jeux de données de test pour tenter de vérifier l'hypothèse 1 143

1.2.1. Cas où le fichier logique est un fichier logique CEC 143

1.2.1.1. Construction de jeu de test pour tenter de vérifier l'hypothèse 2 145

1.2.1.2. Construction de jeux de données de test pour tenter de vérifier l'hypothèse 3. 148

page

1.2.2. Cas où le fichier logique est un fichier logique créancier 150

1.2.2.1. Construction de jeux de données de test pour tenter de montrer que l'hypothèse 4 est vérifiée. 153

2. Construction du second ensemble de jeux de données de test 156

2.1. Construction de jeux de données de test pour tenter de montrer que le programme réalise correctement les contrôles 1 à 3. 156

2.2. Construction de jeux de données de test pour tenter de montrer que l'hypothèse 5 est justifiée. 157

2.3. Construction de jeux de données de test pour tenter de montrer que l'hypothèse 6 est justifiée. 158

Chapitre 4 : Erreurs détectées 160

1. Erreurs détectées dans la première version (Jackson sans inversion) 160

1.1. Erreur de codage 160

1.2. Erreur consécutive à une correction 161

1.3. Oubli d'une instruction 162

2. Erreurs détectées dans la deuxième version (Jackson avec inversion) 162

2.1. Erreur de recopie 162

3. Erreurs détectées dans la troisième version (explicitation des raisonnements) 162

4. Commentaires 163

page

TROISIEME PARTIE : TENTATIVE D'EVALUATION DE LA DEMARCHE PROPOSEE ET COMPARAISON AVEC LES DEMARCHES CLASSIQUES 164

1. Spécifications 164

2. Construction des programmes 165

3. Tests 165

REFERENCES 166

ANNEXES 170

Annexe 1 : Avis de domiciliation. 170

Annexe 2 : Première partie du programme contrôle construit selon la méthode de Jackson (sans inversion) 171

Annexe 3 : Seconde partie du programme contrôle construit selon la méthode de Jackson (sans inversion) 176

Annexe 4 : Programme contrôle construit selon méthode de Jackson en utilisant le principe de l'inversion 182

Annexe 5 : Programme contrôle construit selon la méthode " explicitation des raisonnements" 193

A l'occasion de ce mémoire, je tiens tout d'abord à adresser mes plus vifs remerciements à Baudouin Le Charlier, promoteur de ce travail. Je suis heureuse d'avoir pu bénéficier de ses précieux conseils et je lui suis reconnaissante pour la patience qu'il a témoigné à mon égard.

Je tiens également à remercier le service M.A.I. de son accueil lors de mon stage à la C.G.E.R. J'adresse tout particulièrement mes remerciements à Marie-Claire Verlaine, Reinhilde Van'T Dack, Christian Graas et Herman Segers pour leur collaboration à l'élaboration de la deuxième partie de ce travail.

INTRODUCTION

L'objectif poursuivi à travers ce mémoire est de montrer qu'il est possible et même, à notre avis, plus satisfaisant de concevoir et réaliser une application (de gestion) en utilisant les idées sous-jacentes aux méthodes de démonstration et de construction de programmes. En un mot, l'application de ces idées revient à forcer l'explicitation des raisonnements tout au long du développement d'un projet.
Ces idées sont globalement présentées au paragraphe 1 du chapître 1 de la première partie de ce travail. La façon dont elles doivent être suivies lors de la spécification d'une application, lors de la construction et la "mise au point" des programmes est respectivement précisée aux paragraphes 2, 3 et 4 du même chapître.
Elles ont été appliquées sur un cas réel et les résultats de cette application sont repris dans la 2ème partie de ce travail.
Dans l'intention d'établir une comparaison avec d'autres démarches, nous avons également présenté au chapitre 2 de la première partie des méthodes couramment utilisées pour concevoir et réaliser des applications de gestion. Cette présentation est faite en respectant le schéma suivi lors de la présentation de la première démarche ; dès lors ce chapître 2 se découpe en 3 paragraphes traitant chacun respectivement des méthodes utilisées au niveau de la spécification d'une application, au niveau de la construction des programmes et au niveau des tests de ceux-ci.
Les principes exposés au 2ème paragraphe ont été appliqués sur le cas réel présenté dans la 2ème partie de ce travail et les résultats sont repris au paragraphe 1 du chapître 2 de cette 2ème partie.
Enfin, dans la 3ème partie de ce travail, nous avons tenté de tirer quelques conclusions et notamment d'étudier dans quelle mesure les méthodes présentées dans la 1ère partie peuvent être complémentaires.

P R E M I E R E P A R T I E

DEUX PROPOSITIONS D'AIDE AU DEVELOPPEMENT

DES APPLICATIONS DE GESTION

I N T R O D U C T I O N

La croissance des coûts d'informatique (surtout des coûts de logiciel) et l'apparition dans les annales de l'informatique d'un certain nombre d'échecs retentissants, ont fait prendre conscience petit à petit de l'importance et de la difficulté des premières étapes de développement d'une application, du manque de "méthodes sérieuses" d'aide au développement d'une application.

En réponse à cette prise de conscience, un certain nombre de travaux ont été effectués dans l'espoir de proposer des méthodes, des approches qui aideraient l'analyste-programmeur.

Quoiqu'à l'origine ces travaux aient eu le même objectif, ils se différencient par les moyens mis en oeuvre pour tenter de l'atteindre. Nous relèverons deux classes de travaux.
Dans l'une, on opte pour une solution où l'ordinateur devrait prendre lui-même de plus en plus en charge le travail de l'analyste-programmeur. On propose donc une démarche méthodologique reposant sur l'utilisation de modèles d'outils (automatisés), de règles ; on vise avant tout la rentabilité et le respect des délais.
Dans l'autre, on insiste plus sur les notions de validité, d'explicitation des raisonnements ; on propose des méthodes de démonstration de la validité des programmes.

Dans cette première partie, nous analyserons deux approches d'aide au développement qui nous semblent particulièrement représentatives des deux classes de travaux mentionnées ci-dessus.

Cette analyse se fera à trois niveaux :
au niveau des spécifications, de la construction des programmes et enfin au niveau des tests.

CHAPITRE 1 - PREMIERE PROPOSITION

E X P L I C I T A T I O N D E S R A I S O N N E M E N T S

1. PRESENTATION GENERALE

1.1. Origine : Les méthodes de démonstration de programmes

Les idées de base de la démarche, que nous proposons ci-dessous, trouvent leur origine dans certaines idées propres aux méthodes de démonstration de programmes (voir par exemple LEROY, 78). On y admet que "tout programme doit être tenu pour incorrect" a priori et que les méthodes de validation d'un programme habituellement utilisées (qui consistent "à faire exécuter ce programme par une machine à observer son comportement"..."pour ensuite corriger les erreurs détectées") ne sont pas satisfaisantes. En effet, "l'ensemble des exécutions possibles d'un programme est beaucoup plus vaste que n'importe quel ensemble d'essais qui peuvent être effectués au cours d'une mise au point de durée raisonnable"... par conséquent "les essais ne peuvent suffire à réparer les faiblesses de raisonnement" (LEROY, 75). Aussi, afin de mieux assurer la correction d'un programme ou de mieux s'en approcher, on propose d'expliciter les raisonnements par lesquels on tente de démontrer (au sens mathématique du terme) que le programme est correct. Cette façon de procéder ne permet pas d'obtenir la certitude absolue de la correction du programme car les raisonnements eux-mêmes peuvent contenir des erreurs. Cependant elle offre les avantages suivants :

1) le fait d'essayer de démontrer la correction d'un programme oblige à aller plus loin dans les raisonnements et permet de détecter des erreurs qui seraient restées ignorées sinon ;

2) cette démonstration, pour autant qu'elle soit bien rédigée, constitue, avec ce qui l'accompagne (définitions, spécifications des programmes) la meilleure documentation qui soit : elle permet de mieux comprendre la logique des programmes et de le modifier si nécessaire. (Cet avantage est, à notre avis, le plus important).

1.2. Généralisation des idées sous-jacentes aux méthodes de démonstration de programmes : explicitation de tous les raisonnements

En fait, les méthodes dont nous venons de parler n'ont d'autre but que d'expliciter les raisonnements effectués lors de l'élaboration d'un programme, et par là de les rendre plus rigoureux. Face aux avantages qu' offrent ces méthodes, on peut se demander s'il ne serait pas utile et nécessaire de les appliquer (tout du moins dans leur principe) à d'autres étapes du développement d'une application informatique. Nous pensons plus particulièrement (mais non exclusivement) aux documents produits par les analystes (spécifications, diagrammes de flux, schémas conceptuels...). Ces documents sont le résultat de raisonnements et sont donc tout comme un programme , sujets à l'erreur. D'autre part, généralement, les analystes ne reprennent dans leur dossier que les résultats des raisonnements qu'ils ont effectués. Ils n'éprouvent pas la nécessité de justifier le contenu de leurs dossiers, car ils considèrent celui-ci comme une définition du problème et pour la plupart des gens on ne justifie pas une définition. C'est pour ces raisons, pensons-nous, que ces dossiers sont si difficiles à comprendre et ne constituent pas une bonne documentation.
La solution proposée est alors d'expliciter les raisonnements réalisés à toutes les étapes de l'analyse.

L'idée de cette démarche (qui dépasse le cadre de l'analyse) est donc la suivante : tout document produit lors du développement d'une application doit être justifié tout comme un programme.

En procédant de cette sorte au niveau de l'analyse, on devrait se rendre compte assez tôt de l'adéquation ou de l'inadéquation de l'énoncé du problème tel qu'on l'a formulé et de ses solutions, par rapport à l'énoncé intuitif. En cas d'inadéquation, on procèdera à des corrections jusqu'à ce que l'on soit convaincu (raisonnement à l'appui) que l'énoncé du problème et les solutions retenues correspondent bien au problème réel.

Que ce soit au niveau de l'analyse, de la programmation ou des tests, la démarche proposée consiste à justifier ce que l'on fait. Pour ce, on explicitera les raisonnements qui nous permettent de croire que ce qu'on a construit correspond bien à ce qu'on veut. Il ne sera toutefois pas possible de tout justifier. Cette entreprise serait sans fin. Aussi, on se limitera à justifier les parties qui semblent être les plus délicates et pour lesquelles la validité est particulièrement cruciale. Pour les autres, on se contentera de donner les principales étapes du raisonnement que l'on sera capable de détailler si besoin.

Dans une certaine mesure, nous pouvons comparer ce processus à celui de l'apprentissage d'une langue étrangère : "lorsqu'on étudie une langue étrangère, on accepte de faire des impasses en lisant un roman écrit dans cette langue, car la recherche systématique de toutes les tournures empêcherait d'aller jusqu'au bout ; mais on isole de temps en temps une page témoin, correspondant par exemple à un moment important de l'action, qu'on s'efforce d'analyser en détail et peut-être de traduire, pour se persuader qu'il n'y a pas d'impossibilité théorique à une compréhension totale" (MEYER-BAUDOIN 84, 545 - 546).

Reste évidemment la question de savoir quelles sont les parties délicates pour lesquelles une "démonstration" s'impose. A cette question, on ne peut donner de réponse précise, car tout dépend du problème à traiter.

En fait, l'application correcte des idées exposées ci-dessus, repose avant tout sur la compétence des personnes qui voudraient les mettre en pratique, et c'est peut-être ce qui la distingue fondamentalement de la démarche présentée dans le chapitre 2 se basant sur l'utilisation d'outils automatisables. Elle ne propose pas de règles précises à suivre aveuglément, mais présente quelques idées fondamentales dont il faut tâcher d'acquérir une compréhension suffisante afin d'être capable de voir dans chaque cas comment elles peuvent s'appliquer.

2. SPECIFICATIONS

2.1. Introduction

Nous avons repris, dans cette section, tout ce qui nous semblait être le plus important à préciser au sujet des spécifications (qu'il s'agisse des spécifications de l'application ou des programmes utilisés pour la résoudre). Les idées qui y seront développées sont applicables au delà des premières étapes du développement d'une application. En effet, selon notre point de vue, cette activité qui est celle de spécifier (tout du moins au sens où nous l'entendons) est présente à toutes les étapes du développement d'une application.

Nous allons examiner plus en détail ci-dessous le sens que bon nombre d'analystes-programmeurs (à en juger par les dossiers qu'ils remettent) donnent au concept de spécification et nous le critiquerons. Ceci nous amènera à décrire la démarche de "spécification" que nous avons suivie pour résoudre l'application de gestion décrite dans la deuxième partie de ce travail et nous montrerons en quoi elle nous paraît plus satisfaisante.

2.2. Sens généralement accordé au concept de spécification

On constate assez souvent que les opinions divergent sur la question de savoir ce qu' est une spécification. Nous ne tenterons pas ici de les relever toutes, là n'est pas notre but. Nous nous attacherons plutôt à décrire celle qui nous paraît la plus fréquente chez les analystes-programmeurs.

Le travail de l'analyste consiste à partir de l'énoncé de l'application souvent trop vague et à le préciser. Ce travail l'amènera à accumuler toute une connaissance au sujet du problème qui lui permettra par exemple de trouver des méthodes pour le résoudre, de s'assurer qu'il l'a bien compris ou encore de le faire comprendre à quelqu'un d'autre.

Au terme de ce travail, il produira un ensemble de documents qui reprendront uniquement les résultats auxquels il est arrivé. Selon lui ces documents constitueront la spécification du problème qu'il demandera d'approuver à l'utilisateur.

2.3. Pourquoi l'analyste se contente-t-il de fournir les résultats ?

A cette question, on peut trouver plusieurs réponses. Il est plus facile de communiquer les résultats auxquels on est arrivé que de vouloir expliquer comment on y est arrivé. On a pu, à un moment donné, faire certains choix que l'on n'est pas à même de justifier ou les avoir faits sur base de raisonnements incomplets.

Certains analystes considèrent aussi qu'expliciter les raisonnements est inutile : en effet, pour le programmeur qui devra réaliser l'application, il n'est pas nécessaire de comprendre son utilité réelle vis à vis des utilisateurs, il lui suffit de savoir ce qu'il y a à programmer ; concernant l'utilisateur, l'analyste pense généralement que ses documents ne peuvent être interprétés que d'une seule façon (la sienne) et que l'utilisateur interprétera les choses de la même façon que lui.

D'autres analystes pensent encore qu'il est impossible ou même dénué de sens de justifier les documents résultant de l'analyse parce que l'énoncé de départ est trop vague pour servir de base à une justification.

2.4. Pourquoi cette conception de la spécification n'est-elle pas satisfaisante à notre avis ?

Cette conception de la spécification ne nous semble pas satisfaisante : elle définit une solution mais ne donne pas les moyens, les éléments permettant de vérifier si la solution est acceptable, si l'analyste ne s'est pas trompé dans sa perception du réel. De plus, cette spécification est en général incompréhensible pour l'utilisateur, en tout cas, il est très difficile à celui-ci de se rendre compte si cela correspond à son problème.

2.5. Proposition d'une démarche de spécification

2.5.1. Introduction

Face à ces critiques, nous proposons une autre démarche de spécification basée sur l'idée suivante : tous les raisonnements effectués lors des différentes étapes du développement de l'application devront être explicités, et ce, dans le but de valider les choix retenus et les solutions proposées.

En fait, ces choix et ces raisonnements sont effectués à 3 moments :

1. Lorsqu'on pose le problème, c'est-à-dire lorsque l'analyste rédige un document définissant "ce qu'il y a à faire" et cherche à s'assurer de l'adéquation de ses propositions au problème "réel".
2. Lorsqu'on écrit les programmes.
3. Lorsqu'on réalise les tests.

Aux deux dernières étapes correspondent des formes de raisonnements spécifiques dont nous parlerons plus loin mais elles s'appuient sur la notion de spécification dont nous parlerons ci-dessous.

2.5.2. "Spécification" de l'application ou position du problème

2.5.2.1. En quels termes l'analyste doit-il poser le problème

A ce stade, l'analyste doit s'efforcer de comprendre le problème et d'en fournir une description compréhensible pour l'utilisateur sur laquelle il pourra se mettre d'accord avec celui-ci.
Le point important, selon la démarche proposée ici est que cette description doit avoir un caractère "raisonné". Il sera demandé à l'analyste de justifier pourquoi sa description du problème lui paraît correcte et complète ou éventuellement de fixer les limites de sa validité.
Idéalement, la justification demandée devrait être aussi claire et rigoureuse qu'un bon texte mathématique.
Cette exigence, diront certains, n'est pas raisonnable dans la mesure où le problème à préciser est souvent flou au départ. Cette critique ne nous semble pas fondée : un problème à résoudre n'est flou que dans le sens où il est mal connu et non dans le sens où il serait indéterminé. S'il était flou dans ce second sens, il serait, en effet, impossible de juger la valeur d'une prétendue solution.
Préciser un problème flou doit donc être compris dans le sens : dégager les propriétés importantes d'un problème mal connu mais déterminé. Il doit donc être possible à l'analyste de dégager certains faits réels sur lesquels il pourra baser son argumentation.

2.5.2.2. Sur quoi porte le travail de l'analyste et où s'arrête-t-il ?

Si on considère que poser le problème c'est l'étudier suffisamment pour être capable de décrire et de justifier de manière complète les caractéristiques d'une solution, il est difficile de poser une limite à priori au champ d'investigation de l'analyste.

Nous pensons que la variété des problèmes réels est trop grande pour qu'il soit possible de classifier les difficultés à résoudre et de décomposer l'élaboration d'une application en un certain nombre d'étapes bien définies et immuables (y compris lorsqu'on se "limite" aux applications de gestion). Ceci n'empêche pas que l'on puisse trouver des découpes judicieuses et utiles dans chaque cas particulier (ce qui est non seulement possible mais même nécessaire).

Nous proposons donc de laisser aux personnes chargées de réaliser une application la liberté de choisir la découpe qui leur paraitra adéquate dans le cas particulier sous étude.

2.5.2.3. Enumération d'un certain nombre de tâches ressortant du domaine de l'analyse

Nous pouvons cependant énumérer ici un certain nombre de "tâches ressortant du domaine de l'analyse" qui nous paraîssent pertinentes dans le cas d'application de gestion.

Une des tâches de l'analyste (LE CHARLIER 85, II/18, 2.3.3b) est de comprendre exactement quel est le problème à résoudre. Il faut qu'il puisse valider cette compréhension. Pour cela il présentera à l'utilisateur un document à approuver.
Selon nous, ce document devrait contenir les éléments suivants :

1. L'énumération d'un certain nombre de "faits" supposés vrais par l'analyste, déterminant complètement la solution qu'il envisage, et sur lesquels l'utilisateur pourraît se trouver en désaccord (ceci suppose que tous ces faits soient énoncés dans un langage compréhensible pour l'utilisateur) ;
2. Une justification (valide aux yeux de l'analyste) de l'exhaustivité des faits mentionnés. Le but de ceci est d'éveiller l'attention de l'utilisateur sur les omissions possible (problème de vérifier la complétude).

Dans l'élaboration de l'application elle-même, le travail de l'analyste consiste à choisir des solutions acceptables, des découpes pertinentes. Ces choix seront validés par des raisonnements qui formeront, par la suite, une partie importante de la documentation. Pour que les raisonnements en question puissent être formulés de manière rigoureuse et compréhensible, des concepts et une terminologie adéquats devront être introduits. Pareille démarche devra être appliquée à tous les niveaux de développement de l'application : définition de l'architecture, choix des machines, choix des logiciels, construction des programmes, élaboration des tests,etc...(*)

(*) Les 2 types de tâches dévolues à l'analyste et présentées ci-dessus ne pourront pas nécessairement être menées l'une après l'autre dans l'ordre indiqué. Il se pourrait que l'on doive aller assez loin dans l'étude de choix techniques possibles avant de pouvoir présenter à l'utilisateur une description d'une solution répondant aux exigences énoncées plus haut.

L'énumération ci-dessus n'est pas exhaustive et l'importance à accorder aux différents points mentionnés pourra différer complètement d'une application à l'autre. Par conséquent, il nous paraît peu pertinent d'essayer de préciser dans quel domaine l'analyste devra plus particulièrement faire porter son effort, lors de l'élaboration de toute application.
Nous pensons que le travail d'analyse intervient à tous les niveaux et qu'il est incorrect de vouloir spécialiser à priori (indépendamment d'une application ou d'une organisation particulière) la tâche de l'analyste.

2.5.3. Spécification des programmes

La notion de spécification de programme que nous présentons ici est discutée en détail dans (LE CHARLIER 85, II/10 2.2, II/16 2.3.3.3a). Nous nous contenterons d'en donner une brève présentation.

2.5.3.1. Décomposition d'un problème en sous-problèmes et critère de décomposition

On admet généralement que la réalisation d'une application informatique doit reposer sur une série de décompositions de "problèmes" en sous-problèmes. L'application elle-même constitue le problème initial. On cherche à découvrir (identifier) un certain nombre de sous-problèmes plus simples dont la résolution permettrait de construire aisément une solution du problème initial. Il restera alors à appliquer la même démarche à chaque sous-problème jusqu'à n'avoir plus à résoudre que des problèmes auxquels on puisse directement apporter une solution.

La difficulté essentielle dans la démarche est de savoir ce qu'est une "bonne" décomposition. De nombreux travaux ont été menés dans le but de répondre à cette question (PARNAS, 72a,72b).
A notre avis, le critère essentiel est le suivant : dans une bonne décomposition chaque sous-problème doit posséder un énoncé extrêmement simple.

2.5.3.2. Qu'est-ce qu'un énoncé simple ?

Nous dirons qu'un énoncé est simple, s'il peut servir d'intermédiaire dans un raisonnement convaincant. Ainsi, l'énoncé d'un sous-problème sera simple s'il permet de rendre lisibles et convaincants les raisonnements justifiant la validité de sa solution et celle du problème qui l'utilise.
La simplicité implique la brièveté des énoncés : il serait impossible de construire un raisonnement lisible et convaincant faisant référence à un texte de plusieurs pages. La brièveté n'est cependant pas suffisante. Il faut aussi que les notions utilisées dans l'énoncé soient riches de propriétés connues et bien intégrées par les personnes auxquelles l'énoncé est destiné.
Un énoncé simple ne l'est que pour certaines personnes ayant intégré une certaine "théorie" relative aux concepts utilisés dans l'énoncé. C'est le contenu de la théorie en question qui doit rendre possible les raisonnements utilisant l'énoncé.

2.5.3.3. Les spécifications de programmes sont des énoncés simples (LE CHARLIER, 85, II/16 2.3.3.a)

La décomposition des problèmes en sous-problèmes est utilisée dans bien d'autres disciplines que l'informatique. En mathématique, par exemple, la démonstration d'un théorème difficile s'appuie

toujours sur celles d'un certain nombre de lemmes. Les considérations ci-dessus débordent donc le domaine de l'informatique. Si nous voulons maintenant nous restreindre à ce domaine particulier, on peut considérer qu'un problème (ou un sous-problème) correspond à un programme (sous-programme, module..)(*). L'énoncé du problème sera alors appelé la spécification du programme. On peut, toutefois, donner quelques précisions supplémentaires sur ces énoncés d'un genre particulier.

2.5.3.4. Que doit contenir une spécification de programme (LE CHARLIER, 85, II/59, 2.6)

Selon notre optique, dans une "bonne" architecture, un programme constitue une solution automatisée d'un problème ayant un énoncé simple. Mais les notions figurant dans cet énoncé ne peuvent en général pas être traitées directement par le programme. Celui-ci ne peut manipuler que des représentations de ces notions réalisées au moyen des "objets informatiques" (variables, tableaux, fichiers,...) disponibles dans le langage de programmation. (Par exemple, un programme de salaires et traitements ne traite pas des personnes et des salaires, mais des fichiers contenant des données représentant ces personnes et ces salaires).
Pour ces raisons, la spécification d'un programme doit, en général, se décomposer en deux parties bien distinctes :

- une première partie indiquant quel est le problème résolu par le programme, indépendamment des conventions de représentation ;

(*) Ce point de vue est peut-être trop restrictif et d'autres types de décomposition d'une architecture ont été proposées dans la littérature, voir par exemple (PARNAS, 72a, 72b).

- une seconde partie précisant les conventions de représentation choisies.

C'est la première partie de la spécification qui jouera le rôle essentiel dans les raisonnements, la seconde interviendra surtout dans la justification des interfaces.

2.5.4. Remarque sur les deux sens donnés au mot spécification (LE CHARLIER, 85, II/84, 4.b).

Au paragraphe 2.5.2, spécifier une application (ou plus généralement un problème) signifiait : poser le problème, c'est-à-dire élaborer un texte communicable et précis mettant en lumière toutes les propriétés permettant de le comprendre et de s'atteler à sa résolution.
Au paragraphe 2.5.3., spécifier un programme (ou un problème) signifiait : énoncer le problème, c'est-à-dire en donner une propriété caractéristique simple.

La confusion entre ces deux sens donnés au mot spécification pourrait provoquer des discussions stériles et il serait préférable d'utiliser deux termes distincts (dans (LE CHARLIER, 85) le terme spécification est utilisé uniquement dans le sens "énoncer" le problème ; pour l'autre sens du mot spécification, le terme "théorie" du problème est utilisé). Cette confusion, selon nous, ne se réduit pas à une simple querelle de mots, mais trouve son origine dans certains principes méthodologiques hérités des mathématiques. Il paraît donc important d'insister sur la distinction faite ici, car elle constitue une des originalités de la démarche proposée.

On admet en mathématique qu'on ne peut démontrer la validité d'une propriété sans avoir donné des définitions précises (ou sens mathématique du terme) des objets

concernés par l'énoncé. Si on applique ces idées aux programmes, on en tire que la démonstration de correction d'un programme doit reposer sur une définition précise de ce que doit faire le programme. On débouche sur ce que H. LEROY a appelé le cercle vicieux de la vérité par définition (LEROY, 80). Observons que ce sens donné au mot spécification ne correspond à aucun des deux sens que nous lui avons donnés, bien qu'il puisse aisément être confondu avec l'un ou avec l'autre.

Selon nous, l'idée que la spécification d'un programme est la définition du problème à résoudre ne peut pas servir utilement à appliquer la méthode de résolution d'un problème par découpes successives en sous-problèmes. Car la définition seule ne permet ni de comprendre comment exploiter un sous-problème, ni de comprendre comment le résoudre.
Pour appliquer la méthode en question, il faut acquérir progressivement une compréhension complète et claire du problème (poser le problème) de sorte que chaque partie joue un rôle simple dans sa résolution (possède un énoncé simple).

3. CONSTRUCTION DES PROGRAMMES

De nombreuses méthodes "mathématiques" de démonstration et de construction de programmes sont proposées dans la littérature. Nous ne les développerons pas ici, mentionnons simplement que le lecteur désireux d'en savoir plus à ce sujet pourra se référer à (DIJKSTRA, 76), (LE CHARLIER, 85), (LEROY, 75), (LEROY, 78).

La méthode que nous avons utilisée dans la deuxième partie de ce travail est basée sur la méthode de l'invariant et sur une variante de celle-ci appelée méthode de raisonnement par récurrence ascendante (LEROY, 78). Nous allons en donner une présentation informelle, intuitive ; nous l'illustrerons ensuite par un exemple ; enfin, nous étudierons les problèmes que posent de telles méthodes dans la pratique.

3.1. Présentation de la méthode utilisée dans la deuxième partie

3.1.1. Forme de spécifications

Pour présenter cette méthode, nous supposerons que la spécification d'un programme est définie par une situation initiale et une situation finale. Dans la littérature on parlera de spécification sous forme de pré-postcondition.

La situation initiale explicitera les conditions devant être réalisées pour que l'exécution du programme ait un sens. La situation finale définira (l'effet de) la modification apportée à l'environnement par une exécution du programme pour autant qu'il ait été exécuté dans un environnement vérifiant la situation initiale.

Cette façon de rédiger les spécifications n'est pas forcément toujours la plus pratique mais elle est en tout cas toujours possible. En effet, la

spécification d'un programme est une propriété de la relation liant les données aux résultats et il est toujours possible de formuler cette propriété dans l'énoncé de la situation finale.

3.1.2. Forme des programmes

Le processus de décomposition d'un problème en sous problèmes autorise à admettre que tout programme peut être schématisé par l'une des 2 figures suivantes.

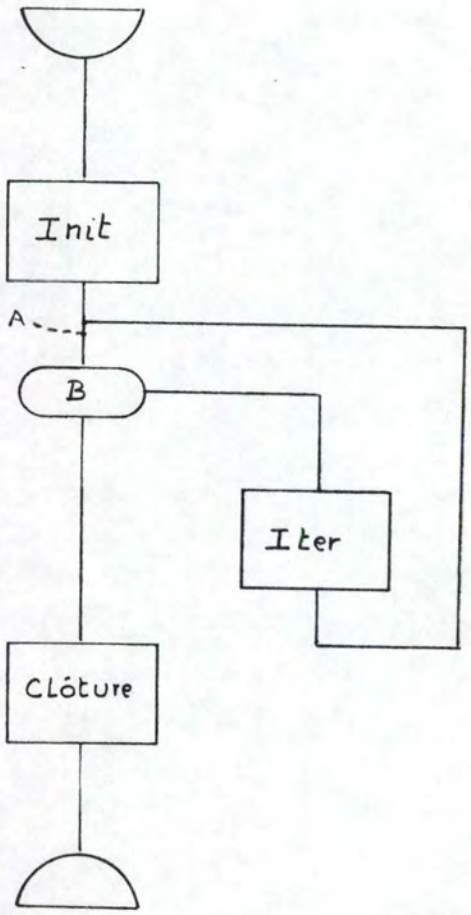


Figure a

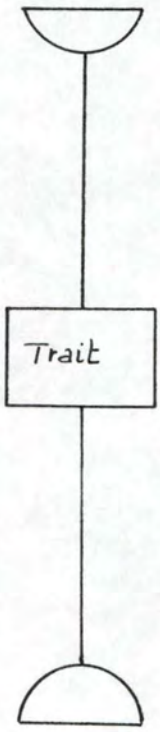


Figure b

Les rectangles figurant dans ces schémas représentent chacun une suite d'instructions, ou plus généralement, une suite de programmes ayant une spécification (simple). On désignera les morceaux de programmes par les noms "Init", "Iter", "Clôture", "Trait" inscrits dans les rectangles.
La lettre "B" désigne une condition, c'est-à-dire un programme de calcul d'une valeur de vérité (vrai ou faux). Les primitives figurant dans ces morceaux de programme sont sensées, a priori, avoir des spécifications.

3.1.3. Principes de la méthode

La justification (ou la construction) d'un programme de structure équivalente à la figure b se fait en montrant que l'exécution successive des instructions constituant ce programme conduit de la situation initiale à la situation finale. Cette justification se fait en exploitant les spécifications des différentes primitives contenues dans ce programme.

La construction d'un programme dont la structure est schématisée par la figure a se déroule selon les étapes suivantes :

1ère étape : <u>Idée intuitive et détermination de la situation générale</u>

On part d'une idée intuitive de solution dans laquelle apparaît une structure répétitive. A partir de là, on essaie de dégager une situation générale, c'est-à-dire un ensemble de propriétés qui seront vérifiées par les variables du programme après un nombre quelconque d'itérations. Cette situation générale exprime quelque chose de commun à toutes les situations particulières (qui seront réalisées après chaque itération). Elle devra être suffisamment précise, c'est-à-dire telle qu'on puisse l'exploiter pour construire le programme selon la méthode proposée.

2ème étape : Détermination des instructions d'initialisation

On suppose la situation initiale vérifiée et on cherche les instructions constituant le morceau de programme "Init", telles qu'après leur exécution, la situation générale soit une première fois vérifiée.

3ème étape : Détermination de la condition d'arrêt et des instructions de clôture

On cherche une condition B à conjuguer à la situation générale pour que la situation finale soit réalisée ou pratiquement réalisée. Ensuite, on suppose la situation générale vérifiée et la condition de fin satisfaite et on cherche les instructions constituant le morceau de programme "clôture" telles qu'après leur exécution la postcondition soit satisfaite.

4ème étape : Détermination des instructions d'itération

On suppose la situation générale vérifiée et la condition de fin non satisfaite, on cherche les instructions constituant le morceau de programme "Iter" telles qu'après leur exécution, la situation générale soit encore vérifiée et qu'on se soit rapproché de la situation finale.

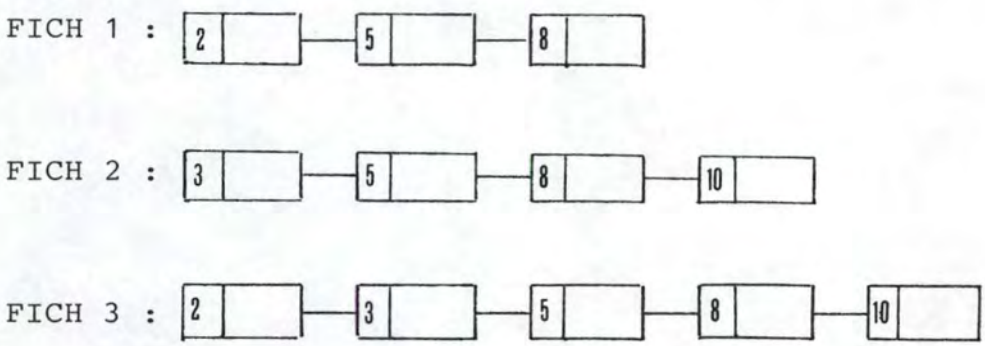
3.2. Illustration

3.2.1. Enoncé

Soient deux fichiers séquentiels d'entrée FICH 1, FICH 2 contenant chacun respectivement n1, n2 articles (n1, n2 $\geqslant$ 0). Chaque article comprend une clé l'identifiant ; deux articles (l'un de FICH 1, l'autre de FICH 2) ayant même valeur de clé ont des contenus identiques. Chaque fichier est trié

en ordre croissant selon les valeurs de ces clés. On demande de constituer un troisième fichier FICH 3 reprenant les articles FICH 1 et FICH 2 tel qu'un article de FICH 1 ayant même valeur de clé qu'un article de FICH 2 ne sera repris qu'une seule fois dans FICH 3. Le fichier FICH 3 comprend n3 articles et est trié en ordre strictement croissant sur les valeurs de ces mêmes clés.

3.2.2. Exemple



3.2.3. Notations

Pour appliquer la méthode décrite ci-dessous, nous allons auparavant réécrire la spécification du problème sous forme de situation initiale, situation finale. Pour ce, nous introduirons les notations suivantes :

a_i désigne le $i^{ème}$ article de FICH 1 ($1 \leqslant i \leqslant n1$)

b_j désigne le $j^{ème}$ article de FICH 2 ($1 \leqslant j \leqslant n2$)

c_k désigne le $k^{ème}$ article de FICH 3 ($1 \leqslant k \leqslant n3$)

$c1_i$ désigne la valeur de la clé de a_i

$c2_j$ désigne la valeur de la clé de b_j

$c3_k$ désigne la valeur de la clé de c_k

3.2.4. Spécification sous forme de situation initiale, situation finale

Situation initiale : FICH 1 contient les articles $a_1, a_2 \ldots a_{n1}$ tels que $c1_1 < c1_2 < \ldots < c1_{n1}$

FICH 2 contient les articles $b_1, b_2 \ldots b_{n2}$ tels que $c2_1 < c2_2 < \ldots < c2_{n2}$

FICH 1, FICH 2 sont fermés.

Situation finale : FICH 1, FICH 2 sont inchangés, FICH 3, FICH 2, FICH 1 sont fermés. FICH 3 contient l'ensemble des articles $c_1, c_2, \ldots c_{n3}$ = $\{a_1, \ldots a_{n1}\} \cup \{b_1, \ldots b_{n2}\}$ tels que

$c3_1 < C3_2 < \ldots < c3_{n3}$

3.2.5. Construction

1ère étape : Idée intuitive et détermination de la situation générale

Pour résoudre ce problème, l'idée intuitive consiste à comparer les valeurs de clés des articles courants a_{i+1}, b_{J+1} de FICH 1 et FICH 2,

si elles sont égales on écrit l'article courant de FICH 1 (ou FICH 2) dans FICH 3 et on passe aux articles suivants dans FICH 1 et FICH 2,

si $c1_{i+1} < c2_{j+1}$, on écrit l'article a_{i+1} dans FICH 3 et on lit l'article suivant dans FICH 1,

si $c1_{i+1} > c2_{j+1}$, on écrit l'article b_{j+1} et on lit l'article suivant dans FICH 2.

A partir de cette méthode intuitive de solution, on essaie de dégager la situation générale. A chaque itération, on aura les fichiers FICH 1, FICH 2 ouverts en lecture, et le fichier FICH 3 ouvert en écriture ; l'ensemble des articles inscrits dans FICH 3 devra être égal à l'union des articles déjà traités dans FICH 1 et FICH 2. Les articles de FICH 3 seront triés en ordre croissant sur les valeurs de leur clé. Toutes les valeurs de clé des articles restant à traiter dans FICH 1 et FICH 2 seront strictement supérieures à toutes les valeurs de clé des articles déjà inscrits dans FICH 3.

Autrement dit, la situation générale sera caractérisée par les 8 conditions suivantes :

(0) $0 \leqslant i \leqslant n_1$, $0 \leqslant j \leqslant n_2$ et $0 \leqslant k \leqslant n_3$ (*)

(1) FICH 1, FICH 2 sont ouverts en lecture.

(2) FICH 3 est ouvert en écriture.

(3) le buffer de FICH 1 contient a_{i+1} si $i < n1$ sinon il contient une valeur de clé ($c1_{n1+1}$) égale à $+\infty$

(4) le buffer de FICH 2 contient b_{j+1} si $j < n2$ sinon il contient une valeur de clé ($C2_{n2+1}$) égale à $+\infty$

(5) le fichier FICH 3 contient l'ensemble des articles $\{c_l \,/\, 1 \leqslant l \leqslant k\} = \{a_m \,/\, 1 \leqslant m \leqslant i\} \cup \{b_n \,/\, 1 \leqslant n \leqslant j\}$

(6) pour $1 \leqslant l \leqslant k$ on a $c3_l < c3_{l+1}$

(7) $\min(\{c1_{i+1}, c1_{2+2} \ldots c1_{n1+1}\} \cup \{c2_{j+1}, c2_{j+2} \ldots c2_{n2+1}\}) > \max\{c3_l \,/\, 1 \leqslant l \leqslant k\}$

(*) i, j, k, ne sont pas des variables du programme, mais représentent des valeurs indéterminées dépendant du nombre d'itérations effectuées. Pour être rigoureux, il aurait fallu les quantifier en écrivant "il existe i, j, k tels que les conditions 0 à 7 sont vérifiées".

Remarque sur la sémantique des instructions utilisées

Les instructions d'ouverture, de fermeture, de lecture et d'écriture que nous utilisons ont une sémantique semblable à celle de Cobol.
Ainsi, l'ouverture d'un fichier en lecture a pour effet de rendre courant le premier article s'il existe ou de déclencher la condition de fin de fichier sinon. L'ouverture d'un fichier en écriture crée un fichier vide.
L'opération de lecture a pour effet de mettre l'article courant dans le buffer et de rendre l'article suivant courant s'il existe, sinon la condition de fin est déclenchée. Si la condition de fin est déclenchée, l'instruction de lecture a pour effet de mettre $+\infty$ dans la zone du buffer contenant la clé.
L'opération d'écriture a pour effet de rajouter en fin de fichier l'enregistrement contenu dans le buffer.

2ème étape : Recherche des instructions d'initialisation

Les instructions d'initialisation sont :

OUVRIR FICH 1, FICH 2	en lecture	(I_1)
OUVRIR FICH 3	en écriture	(I_2)
LIRE FICH 1		(I_3)
LIRE FICH 2		(I_4)

Justification : montrons qu'après l'exécution des instructions I_1, I_2, I_3, I_4 les conditions (0) à (7) sont vérifiées avec (i, j et k = 0).

(0) : OK puisque i, j, et k = 0

(1) : OK puisqu'après I_1 les fichiers FICH 1 et FICH 2 sont ouverts en lecture.

(2) : OK car après I_2, le fichier FICH 3 est ouvert en écriture.

(3) : OK car après I_3, le buffer de FICH 1 contient le premier article de FICH 1, c'est-à-dire $a_1 = a_{i+1}$ (avec $i = 0$) si $i < n_1$ sinon il contient une valeur de clé égale à $+\infty$

(4) : OK après I_4, le buffer de FICH 2 contient le premier article de FICH 2, c'est-à-dire $b_1 = b_{J+1}$ (avec $j = 0$) si $j < n_2$ sinon il contient une valeur de clé égale à $+\infty$

(5) : OK car avec i, j, k = 0 les ensembles $\{c_l / 1 \leqslant l \leqslant k\}$, $\{a_m / 1 \leqslant m \leqslant i\}$ et $\{b_m / 1 \leqslant n \leqslant j\}$ sont tous vides et $\emptyset = \emptyset \cup \emptyset$.

(6) : OK car k = 0 et l'intervalle 1,k est vide

(7) : OK car $\min(\{c1_1, c1_2 \ldots c1_{n1+1}\} \cup \{c2_1 \ldots c2_{n2+1}\}) = x$ avec $x \in]-\infty, +\infty[$ et $\max\{c3_l / 1 \leqslant l \leqslant k\} = \max \emptyset = -\infty$

3ème étape : <u>Recherche de la condition de fin et des instructions de clôture</u>.

On suppose la situation générale vérifiée et notons CLE 1, CLE 2 les zones des buffers des fichiers FICH 1, FICH 2 contenant les clés.
Le fichier FICH 3 aura un contenu correspondant à la situation finale lorsque i = n1 et j = n2 (d'après (5), (6) et définition de n3).
Ces conditions sont équivalentes à CLE 1 = $+\infty$ et CLE 2 = $+\infty$ d'après (3) et (4) ; pour que la postcondition soit satisfaite, il ne reste plus qu'à fermer les 3 fichiers FICH 1 , FICH 2, FICH 3.

F E R M E R FICH 2, FICH 2, FICH 3

4ème étape : <u>Recherche des instructions d'itération (Iter).</u>

On suppose que la condition de fin d'itération n'est pas remplie et que la situation générale est vérifiée. On cherche les instructions telles qu'après leur exécution on soit toujours dans la situation générale tout en s'étant rapproché de la situation finale, soient :

si CLE 1 < CLE 2	C1
alors écrire le buffer de FICH 1 dans FICH 3	I1
lire FICH 1	I2
sinon si CLE 1 > CLE 2	C2
alors écrire le buffer de FICH 2 dans FICH 3	I3
lire FICH 2	I4
sinon écire le buffer de FICH 1 dans FICH 3	I5
lire FICH 1	I6
lire FICH 2	I7

Justification : nous distinguons trois cas selon que la valeur de la clé de l'article courant de FICH 1 est soit inférieure, soit supérieure, soit égale à celle de l'article courant de FICH 2.

- 1er cas : la valeur de la clé de l'article courant de FICH 1 est inférieure à celle de l'article courant de FICH 2 ; la condition C1 est vraie et les instructions I1, I2 seront exécutées. Après leur exécution, la situation générale sera encore vérifiée avec de nouvelles valeurs i', j', k' égales respectivement à i+1, j, k+1.
  En effet :
  - la condition (0) est vérifiée puisque CLE 1 < CLE 2 on a $i+1 \leqslant n_1$ donc $i' \leqslant n_1$
  - les conditions (1) et (2) seront vérifiées puisqu'elles l'étaient avant l'exécution de I1, I2 et que l'exécution de ces instructions ne les modifie pas

- la condition (3) est remplie puisqu'après I2 le buffer de FICH 1 contient le i'+1ème article de FICH 1, c'est-à-dire a_{i+2} si $I+2 \leqslant n_1$ sinon il contiendra une valeur de clé égale à $+\infty$
- la condition (4) sera satisfaite puisqu'elle l'était avant l'itération et que les instructions I_1, I_2 n'ont aucun effet sur le contenu du buffer de FICH 2.
- la condition (5) est vérifiée, en effet, avant l'exécution de I_1, I_2 on avait

  $\{c_l/1 \leqslant l \leqslant k\} = \{a_m/1 \leqslant m \leqslant i\} \cup \{b_n/1 \leqslant n \leqslant j\}$

  après I_1, I_2 on aura traité un élément supplémentaire dans FICH 1 à savoir l'élément a_{i+1} et on l'aura écrit dans FICH 3, c'est-à-dire

  $\{c_l/1 \leqslant l \leqslant k\} \cup \{a_{i+1}\} = \{a_m/1 \leqslant m \leqslant i+1\}$
  $\cup \{b_n/1 \leqslant n \leqslant j\}$

  $\{c_l/1 \leqslant l \leqslant k+1\} = \{a_m/1 \leqslant m \leqslant i+1\} \cup \{b_n/1 \leqslant n \leqslant j\}$

  $\{c_l/1 \leqslant l \leqslant k'\} = \{a_m/1 \leqslant m \leqslant i'\} \cup \{b_n/1 \leqslant n \leqslant j\}$

  avec
  $k' = k+1$
  $i' = i+1$
  la condition (5) sera vérifiée pour i', j, k'.

- la condition (6) sera satisfaite puisqu'après exécution de I_1, I_2, on a :
  * $c31 < \ldots < c3k$ (car vrai avant l'itération)
  * $c3K+1 = c1_{i+1}$

  mais puisqu'on avait (7) avant itération on a :

  $$c3k+1 = c1_{i+1} \geqslant \min\left(\{c1_{i+1} \ldots c1_{n1+1}\} \cup \{c2_{j+1} \ldots c2_{n2+1}\}\right)$$
  $$> \max\{c3_l/1 \leqslant l \leqslant k\}$$

  d'où $c3_1 < \ldots c3_k < c3_{k+1}$

- la condition (7) sera vérifiée car on aura

$$\max\{c3_l / 1 \leqslant l \leqslant k+1\} = c3_{k+1}$$
$$= c1_{i+1} < \min(\{c1_{i+2},\ldots,c1_{n1+1}\} \cup \{c2_{j+1},\ldots,c2_{n2+1}\})$$

(car $c1_{i+1} < C1_{i+2}$ et $C1_{i+1} = CLE1 < C2_{j+1} = CLE2$).

D'où $\max\{C3_l / l < k'\} < \min(\{c1_{i+1}\ldots, c1_{n1+1}\} \cup \{C2_{j+1},\ldots,c2_{n2+1}\})$ (car $k' = k+1$ et $i' = i+1$).

- le 2ème cas (CLE1 > CLE2) se traite de manière symétrique.

- le 3ème cas ($CLE1 = CLE2 \neq +\infty$) peut être justifié par un raisonnement légèrement différent que nous omettons pour être concis.

3.3. Applicabilité pratique de la méthode proposée

L'applicabilité de la méthode proposée dans la pratique courante est un sujet délicat et controversé.
En laissant même de côté le fait que peu de programmeurs actuels sont prêts à se plier à l'exigence de rigueur demandée par de telles méthodes, il reste que même des partisans d'une méthodologie rigoureuse estiment qu'elles sont trop difficiles ou trop longues à appliquer dans la réalité(voir par exemple (MEYER-BAUDOIN, 84, p 544 - 546)).

Nous ne partageons pas cette opinion, parce qu'elle repose selon nous sur une optique trop formaliste de la façon d'utiliser les méthodes en question. A notre avis, leur but est de rendre explicites et communicables les raisonnements que l'on doit faire, de toute façon, lorsqu'on

construit un programme si l'on veut qu'il ait de "bonnes chances" d'être correct.
La vraie difficulté, qu'il ne faut pas essayer de minimiser, c'est de choisir un langage adéquat et le niveau de détail le mieux adapté lorsqu'on énonce les raisonnements. Si l'on raisonne avec trop de détails les raisonnements seront inextricables et impossibles à comprendre. Si l'on raisonne avec trop peu de détails, on risque d'oublier certains cas. La seule "solution" à ce "problème", c'est d'admettre que la construction de programmes corrects et bien documentés demande de grandes qualités de rigueur et de clarté.

4. TESTS

4.1. Raison d'être des tests

Dans la pratique courante, les tests sont souvent la seule manière de "vérifier" la correction des programmes. Cette importance accordée aux tests est due au peu de confiance que l'on a dans la validité des raisonnements faits lors de la construction des programmes. Mais si on peut douter de la validité de ces raisonnements, c'est surtout parce qu'ils sont rarement énoncés rigoureusement et explicitement.

Nous pensons (et cette opinion est confirmée par l'expérience) que l'utilisation des méthodes de raisonnement proposées plus haut rend beaucoup moins nécessaire le recours aux tests,sans toutefois les rendre totalement inutiles. En effet, les erreurs de raisonnements ne peuvent jamais être absolument exclues quel que soit le soin qu'on y ait apporté ; et des erreurs matérielles comme des fautes de recopie restent toujours possibles.

Nous considérons surtout les tests comme un moyen de détecter les erreurs accidentelles dues non à une lacune importante du raisonnement mais à une non conformité (du programme réellement écrit) à celui-ci (exemple : on construit un programme qui utilise une variable i pour parcourir un tableau, mais suite à une erreur de recopie, cette variable est remplacée par une autre, dans une instruction).

Nous croyons également que ce genre d'erreurs ne peut généralement pas être détecté par autre chose que des tests. La classe des erreurs détectables automatiquement se réduit pratiquement aux erreurs syntaxiques (exemple : variable non déclarée,...) comme le montre la théorie de la calculabilité (LEROY, 85).

4.2. Principe de la construction des tests

Comme DIJKSTRA l'a fait remarquer dans (DIJKSTRA, 76) il est impossible de prouver la correction d'un programme au moyen de tests parce que l'ensemble des cas à tester sera toujours infini ou pratiquement infini. Le choix d'un jeu de tests pris "au hasard" est manifestement peu satisfaisant.

Nous proposons de construire les jeux de tests par des raisonnements semblables à ceux faits pour construire les programmes. Pour cela, nous partirons de l'hypothèse de base que le programmeur n'a pas essayé d'écrire un programme faux (spécialement conçu pour déjouer les tests et provoquer des erreurs dans des cas expressément choisis) mais qu'au contraire, il a essayé de construire un programme correct en utilisant les méthodes les plus simples et les plus générales.
On construira alors des tests en formulant une série d'hypothèses sur la manière dont le programme aura été construit, de telle façon qu'on puisse pratiquement démontrer qu'un programme respectant les hypothèses et réagissant correctement aux tests doive nécessairement être correct.

4.3. Illustration

Par exemple, si un programme est supposé calculer la somme d'une suite de nombres, on supposera qu'il ne contient qu'une boucle et que tous les éléments de la suite sont traités de la même façon (i.e par la même suite d'instructions) au moins à partir du deuxième. Il suffira donc de le tester avec la suite vide, des suites à un élément et des suites à deux éléments. Il faudra encore choisir les éléments de ces suites de manière suffisamment générale.

4.3.1. Cas des suites à un élément

On supposera que l'exécution ne dépend pas du signe de l'unique élément x de la suite et qu'il produit un résultat qui est fonction linéaire de celui-ci. Le résultat est donc de la forme $\alpha x + \beta$ et est correct si et seulement si $\alpha = 1$ et $\beta = 0$.

Pour vérifier cette condition, il suffit d'exécuter ce programme successivement avec x = 0 et x = 1.

En effet, si le programme fournit le résultat 0 pour x = 0 on a $\alpha 0 + \beta = 0$ d'où $\beta = 0$.

Ensuite si le programme fournit le résultat 1 pour x = 1 on a $\alpha 1 + \beta = 1$ d'où $\alpha = 1$.

4.3.2. Cas des suites à 2 éléments

On supposera que l'exécution ne dépend pas du signe des deux éléments x et y de la suite et qu'il produit un résultat de la forme $\alpha x + \beta y + \gamma$ il est correct si et seulement si $\alpha = 1$, $\beta = 1$ et $\gamma = 0$.

Pour vérifier cette condition, on exécutera le programme successivement avec x = 0 et y = 0 puis avec x = 1 et y = 0 et finalement avec x = 0 et y = 1.

En effet, si le programme fournit le résultat 0 pour x = 0 et y = 0 on a $\alpha .0 + \beta .0 + \gamma = 0$ => $\gamma = 0$.

Ensuite si le programme fournit le résultat 1 pour x = 1 et y = 0 on a $\alpha .1 + \beta .0 + \gamma = 1 \Rightarrow \alpha = 1$.

Finalement si le programme fournit le résultat 1 pour x = 0 et y = 1 on a $\alpha .0 + \beta .0 + \gamma = 1$ => $\beta = 1$.

On obtient donc en tout un jeu de 6 tests :
- la suite vide, la suite réduite à l'élément 0, la suite réduite à l'élément 1, les suites à 2 éléments dont les valeurs sont respectivement (0,0), (0,1) et (1,0).

4.4. Avantages de la méthode proposée

Non seulement la méthode proposée ci-dessus permet de construire les jeux de test de manière plus rigoureuse mais elle fournit aussi une aide précieuse en cas de découverte ultérieure d'une erreur dans le programme (lors de son exploitation), car on pourra parcourir la liste des hypothèses faites sur le programme, rechercher celle(s) qui est (sont) la (les) plus probablement mise(s) en défaut et ainsi découvrir plus rapidement la lacune du programme.

4.5. Détermination des hypothèses

Il est difficile de dire en toute généralité comment choisir les hypothèses à faire sur le programme à tester. Celles-ci ne pourront être déterminées que sur base du problème particulier. Quelques principes généraux peuvent cependant être énoncés basés sur la notion de décomposition en sous-problèmes et sur la méthode de construction de programme utilisée.

4.5.1. Décomposition en sous-problèmes

Si un programme P utilise les programmes (sous-problèmes) P1...Pn, on fera l'hypothèse que P1, ..., Pn sont corrects lors du choix des tests de P.

P1,...Pn seront alors testés séparément. S'il est trop difficile de les tester individuellement, on construira des jeux de tests pour chacun d'eux sous l'hypothèse que P les utilise correctement. Exemple : voir 2ème partie.

4.5.2. Forme des programmes

La construction des programmes par la méthode proposée plus haut conduit à des programmes de forme bien déterminée. On pourra en déduire certaines hypothèses sur la façon dont un programme traite un problème donné.

* * *

CHAPITRE 2 - SECONDE PROPOSITION

APPROCHE ORIENTEE MODELES - OUTILS - REGLES

1. INTRODUCTION

Dans ce deuxième chapitre, nous analyserons trois méthodes d'aide au développement d'application de gestion. Chacune d'elles correspond à une des étapes de ce développement (phase de spécification, phase de construction de programmes, phase de test).
Provenant de différentes démarches méthodologiques, ces trois méthodes se recoupent de temps en temps ou présentent certaines incompatibilités. Elles peuvent cependant, moyennant un certain travail d'adaptation, constituer une démarche méthodologique représentative de l'approche actuelle des problèmes de gestion.

2. UNE METHODE DE SPECIFICATION DES APPLICATIONS INFORMATIQUES DE GESTION : L'ANALYSE FONCTIONNELLE

2.1. Introduction

Nous présenterons succinctement la méthode d'analyse fonctionnelle développée à l'institut d'informatique de Namur. Celle-ci a pour objectif premier la spécification et la validation d'applications de gestion de "grande taille" (systèmes d'information). Ses concepts les plus fondamentaux sont toutefois utiles à la spécifications d'applications de moindre importance. C'est sur ceux-ci que nous insisterons

davantage.
Dans un premier paragraphe (2.2.), nous présenterons la notion de "schéma conceptuel" : résultat de la phase de spécification et point de départ de l'implémentation. Le deuxième paragraphe (2.3.) traitera des objectifs poursuivis au cours de cette phase : à quoi sert le schéma conceptuel ? En quoi permet-il de faciliter et de rendre plus fiable la réalisation de l'application ? Le troisième paragraphe (2.4.) décrira brièvement les modèles et les outils proposés par la méthode. Enfin dans le dernier paragraphe (2.5.), nous ferons quelques commentaires sur cette méthode de spécification.

2.2. La notion de schéma conceptuel

Le point de départ de la méthode de spécification proposé par F. Bodart et Y. Pigneur dans (Bod. Pig. 83) est la notion de système d'information (S.I.). Le fonctionnement d'une entreprise peut être vu comme la réalisation dans le temps, d'un certain nombre de traitements, nécessitant une certaine quantité de ressources. L'informatique s'intéresse aux traitements portant sur des informations (ou faits), suffisamment structurés pour être, au moins en principe, automatisés. On appelle système d'information d'une organisation l'ensemble formé de ces traitements, des informations qui s'y rapportent et des ressources qu'ils utilisent.

Le but de la phase de spécification (ou analyse fonctionnelle) est de produire un schéma conceptuel du S.I., c'est-à-dire une description des informations, des traitements et des ressources constituant celui-ci. Cette description sera réalisée à l'aide d'un langage formel (DSL : Dynamic Specification Language). Un texte écrit en DSL définit plusieurs modèles du S.I. représentant les différents aspects de celui-ci. Certains aspects ne pouvant être pris en compte par les modèles (tel l'effet exact des traitements) seront exprimés en langage naturel, sous forme de commentaires.

2.3. Utilité de l'analyse fonctionnelle - difficultés

La construction d'un schéma conceptuel doit permettre de réaliser les objectifs suivants :

1) fournir une définition précise du S.I. pouvant servir de base sûre à l'implémentation,
2) s'assurer à priori de la conformité du futur S.I. aux besoins réels,
3) vérifier la faisabilité de celui-ci.

On veut donc définir de manière précise ce qu'il y a à faire, sans se préoccuper de comment on va le faire, mais en étant capable de vérifier qu'on pourra le faire et qu'on obtiendra un système réellement satisfaisant. Ces objectifs comportent visiblement de réelles difficultés. Par exemple, jusqu'à quel niveau de détail faut-il aller dans la description du S.I., comment vérifier la conformité aux besoins, sans "essayer" le S.I. dans la réalité et observer les réactions des utilisateurs ?, peut-on vérifier totalement la faisabilité du S.I. sans le construire réellement ?

Il n'est pas possible de trouver dans (BOD.PIG. 83) des réponses absolument satisfaisantes à ces questions.
Dans l'esprit des auteurs sans doute vaut-il mieux proposer des solutions partielles (de préférence automatisables) que de laisser pleine liberté à l'analyste.

2.4. Les modèles et les outils

2.4.1. Le modèle ENTITE - ASSOCIATION (description des "données")

2.4.1.1. Remarques préliminaires

Le modèle ENTITE-ASSOCIATION propose un certain nombre de concepts permettant de décrire les données mémorisées dans le système d'information. Ces concepts ont été choisis de manière à permettre une description "structurée" de ces données adaptée à la

fois à la compréhension humaine et au passage ultérieur à l'implémentation sous forme de fichiers ou de base de données.
Avant de présenter ces concepts, il nous paraît utile de faire quelques remarques préliminaires.

1. La distinction entre information (fait réel) et donnée (représentation de l'information sous forme de caractères, de nombres ou autre) n'est pas faite clairement dans (BOD.PIG. 83).

   A notre avis, le modèle Entité-Association doit être considéré comme un modèle de description des données, non des informations, et nous le présenterons comme tel.

2. En fait, il existe de nombreuses variantes et "améliorations" de ce modèle. On se contentera, ici, de donner une description rapide de ses concepts les plus fondamentaux.

2.4.1.2. Contenu conceptuel de la base des données : entités, associations et valeurs

Le contenu de la base des données du système d'information, à un instant donné, sera "vu" comme un ensemble fini d'objets appelés entités reliés par des associations et qualifiés par des valeurs. Nous appellerons cette "vue" : contenu conceptuel ou état conceptuel de la base des données à l'instant t.

Une entité est un objet de nature indéterminée au contraire d'une valeur qui est un objet bien précis comme un nombre ou une chaîne de caractères.

2.4.1.3. Structuration de la base des données : types d'entités, types d'associations et propriétés

Les entités et les associations sont regroupées en classes appelées types d'entités et d'associations.

(On utilise aussi souvent les termes "entité" et "association" au lieu de "type d'entités" et "type d'associations" ; on parle, alors d"bccurnences d'entité ou d'association" au lieu d'"entité" ou d'"association". Le mélange des deux terminologies peut conduire à des confusions et nous nous tiendrons, ici, à la première).

Intuitivement, les entités d'un même type représentent des objets du monde réel de "même nature". De même, des associations d'un même type représentent des relations de "même nature" entre objets semblables. En fait, la signification exacte d'un type d'entité ou d'association doit être précisée par une convention explicite car le modèle Entité Association ne permet pas d'exprimer cette signification.

La définition d'un type d'entité comporte son nom et la liste des propriétés qui lui sont attachées. Une propriété est définie par un nom et un ensemble de valeurs. Le fait que P soit une propriété attachée au type d'entité E signifie que toute entité e de type E doit être caractérisée par une et une seule valeur v de la propriété P.
La notion de propriété peut être raffinée de plusieurs façons. Par exemple, on pourra considérer des propriétés identifiantes (il y a une bijection entre les entités et les valeurs qui les caractérisent); des propriétés facultatives (la valeur caractérisant l'entité peut être omise), des propriétés répétitives (les entités sont caractérisées par une liste ou un ensemble de valeurs de la propriété).

La définition d'un type d'association comporte son nom, la liste des types d'entités qui y participent et la liste des propriétés qui lui sont attachées. Si $E_1,\ldots,E_n$ est la liste des types d'entités qui

participent au type d'association A, une association de type A est un n-uple $(e_1 \ldots, e_n)$ d'entités des types respectifs $E_1, \ldots, E_n$.
Des valeurs de propriétés sont attachées aux associations de la même façon qu'aux entités.

2.4.1.4. Contraintes d'intégrité

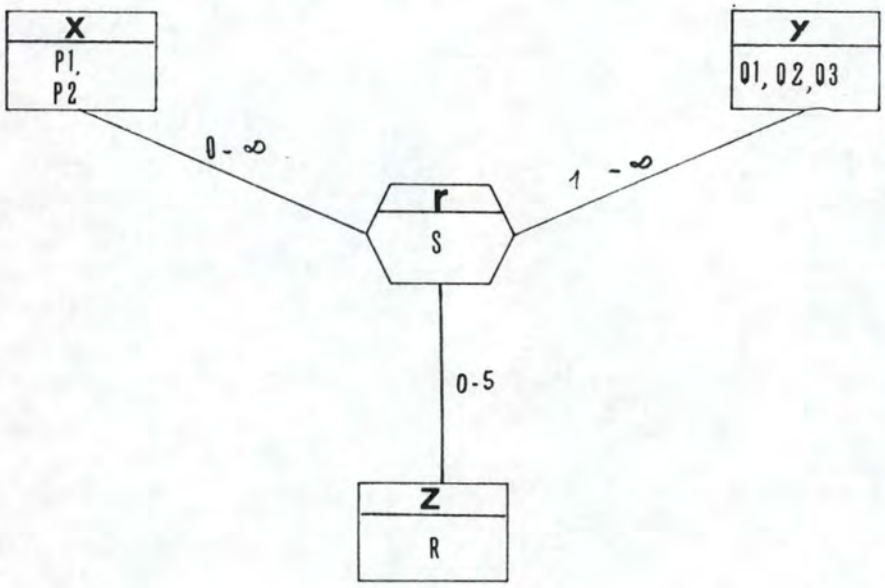
Il est possible d'imposer au contenu conceptuel de la base de données des contraintes d'intégrité, c'est-à-dire des conditions à respecter à tout instant.
Ces contraintes d'intégrité peuvent être plus ou moins complexes selon la version choisie du modèle Entité-Association. La plus utilisée est celle de connectivité d'un type d'association qui consiste à fixer, pour tout type d'entités E participant à un type d'association A, le nombre minimum et le nombre maximum de fois qu'une entité e du type E doit (peut) figurer dans une association de type A.

2.4.1.5. Définition des contenus conceptuels possibles de la base de donnée : le schéma Entité-Association

Le schéma Entité-Association de la base de données du Système d'Information définit l'ensemble des contenus conceptuels possibles de celle-ci. Il est constitué de la définition des types d'entités et des types d'associations de la base de données ainsi que des contraintes d'intégrité qui s'y appliquent.

2.4.1.6. Description graphique

Le modèle Entité-Association propose, pour représenter un schéma, un symbolisme dans lequel les types d'entités sont représentés par des rectangles contenant leur nom et ceux de leurs propriétés. Les types d'associations sont représentés par des hexagones contenant leur nom et ceux de leurs propriétés, reliés par un trait à chaque rectangle représentant un type d'entités participant à l'association. Chaque trait est accompagné de deux nombres précisant la connectivité de l'association. Ainsi, le diagramme :



représente les types d'entités X (de propriétés p1, p2), Y (de propriétés q1, q2, q3) et Z (de propriété r) participant au type d'associations R (de propriété s). Il exprime, de plus, qu'une entité de type Y doit participer à une association de type R, au moins et qu'une entité de type Z ne peut participer à plus de 5 associations de ce type.

2.4.2. Les modèles de traitements

Trois modèles sont proposés. Chacun d'eux s'attache à décrire les traitements vus sous un angle particulier. Comment les traitements se décomposent-ils ? (structuration des traitements). Comment s'enchainent-ils (dynamique des traitements) ? Quels sont leurs effets (statique des traitements) ?
Nous donnerons ci-dessous une brève présentation de ces 3 modèles.

2.4.2.1. Le modèle de structuration des traitements

Le modèle de structuration des traitements permet de décrire la décomposition d'un projet en traitements de plus en plus fins. Il fournit des critères permettant de décomposer un projet en traitements ainsi qu'un certain nombre de concepts associés à chacun des niveaux privilégiés lors de cette décomposition.
Les concepts de ce modèle sont les suivants :

- le concept de projet qui correspond à la mise en oeuvre du S.I. tout entier,
- le concept d'application qui est un traitement "quasi autonome par rapport aux autres applications d'un projet",
- le concept de phase qui est un traitement "manuel ou automatisable réalisé sans interruption, sans changement de lieu et ni de ressources",
- le concept de fonction qui correspond à un traitement de niveau élémentaire.

Les critères d'identification de ces concepts résultent de leur définition. Ils manquent parfois de précision et ne sont donc pas toujours faciles à utiliser. Ils doivent être considérés comme guides et non comme des règles à suivre aveuglément. Le lecteur désireux d'en savoir plus à ce sujet se réfèrera à (BOD.PIG. 83).

La description d'un traitement reprend le nom de ce traitement, son niveau (projet, application, phase, fonction) et les traitements dont il est éventuellement un composant.

2.4.2.2. Le modèle de la dynamique des traitements

Le modèle de la dynamique sert à décrire les conditions de déclenchement, d'exécution et d'enchainement des traitements. A proprement parler il n'est pas utile à la spécification du S.I. mais constitue l'élément de base nécessaire à la vérification du caractère faisable du schéma conceptuel (via un outil de simulation). Ce modèle repose sur les concepts de processus, d'événement et de structures dynamiques élémentaires :

- un processus est l'exécution d'une procédure de traitement de l'information,
- un événement correspond à un changement d'état du S.I.

On distingue 4 types principaux d'événements : l'activation, la terminaison d'un processus, l'envoi, la réception d'un message (un message est un échange d'information entre 2 processus ou entre un processus et l'environnement du S.I.).

L'enchainement des traitements est exprimé par combinaison des primitives dynamiques (telles structures d'enchainement séquentiel, conditionnel, convergent...). Ainsi, pour exprimer que la terminaison d'un processus provoque le déclenchement d'un autre processus, on utilisera le concept d'enchainement séquentiel.

- Les 2 modèles présentés ci-dessus ne sont pas indépendants ; la dynamique suppose que l'on ait d'abord réalisé la structuration des traitements.

2.4.2.3. Le modèle de la statique des traitements

Le modèle de la statique sert à décrire l'effet des traitements. Cette description est réalisée en spécifiant d'une part des messages-données et les informations du S.I. nécessaires à l'obtention des messages-résultats, et d'autre part en spécifiant les règles de traitement qui assurent l'obtention des messages-résultats.
Ces règles de traitement sont énoncées en langue naturelle.

2.4.3. Le modèle des ressources

Le modèle des ressources sert à caractériser les ressources nécessaires à l'exécution des traitements. Ce modèle constitue avec celui de la dynamique les éléments de base nécessaires à l'évaluation du caractère réalisable du schéma conceptuel.
Les concepts qu'il fournit sont ceux de ressource réutilisable et de ressource consommable. La description d'une ressource (réutilisable ou consommable) comprend notamment le nom de cette ressource, son unité de mesure, sa capacité (nombre d'unités disponibles), son calendrier de disponibilité, les traitements qui la requièrent.

2.4.4. Les outils automatisés

Un certain nombre d'outils sont proposés dans le but de vérifier la validité du schéma conceptuel et de le documenter de manière automatisée. Ces outils font partie du système logiciel I.D.A. (Interactive Design Approach) réalisé à l'institut d'informatique des Facultés Universitaires de Namur, en coopération avec le projet I.S.D.O.S.. Le coeur de ce système est la base de données des spécifications. Elle reprend l'ensemble des spécifications du S.I. exprimées en

D.S.L. (Dynamic Specification Language) et constitue la source unique de toute la documentation.
Elle supporte une large variété d'outils qui ont quatre fonctions principales :

- génération de rapports documentaires présentant sous des formes variées (listes, tableaux...) différents aspects du système spécifié,
- contrôle de cohérence et de complétude,
- simulation du "fonctionnement" du S.I. pour évaluer le caractère réalisable moyennant les ressources disponibles,
- génération d'une maquette programmée du futur système pour tester le caractère effectif des spécifications (conformité aux besoins).

L'intérêt de ces outils est qu'ils permettent d'introduire une approche plus expérimentale au niveau de l'analyse fonctionnelle, d'une part en testant l'impact de diverses hypothèses de fonctionnement et d'autre part en favorisant la perception directe par les utilisateurs de ce qu'ils souhaitent obtenir.

2.5. Commentaires

Pour clore ce paragraphe, il nous a semblé bon de faire quelques remarques sur la méthode de spécification présentée ci-dessus. En fait, ce qui est proposé nous semble être davantage un outil de validation que de spécification. Nous sommes arrivés à cette conclusion en examinant ce que cette méthode offrait d'une part pour spécifier une application et d'autre part pour valider cette spécification.

2.5.1. Outil de spécification

Concernant le travail de spécification proprement dit (au sens : poser le problème) on ne dispose que d'un outil (modèle) de spécification des données (non des informations) ; pour spécifier les traitements aucune règle méthodologique n'est proposée. La plupart des modèles fournis par la méthode ne sont réellement utiles que pour valider le schéma conceptuel.

2.5.2. Outil de validation

La validation d'un schéma conceptuel consiste à vérifier s'il est, d'une part, conforme aux besoins et, d'autre part, réalisable moyennant les ressources disponibles.

- Pour tester la conformité aux besoins, on dispose : du modèle entité-association (présenté par les auteurs comme un langage clair favorisant la discussion avec l'utilisateur) et d'outils automatisés permettant de détecter les lacunes (incohérence, non conformité...) mais non de vérifier de manière absolue la conformité aux besoins.

- Pour tester le caractère réalisable du S.I., on dispose d'un outil de simulation. Cet outil permet de s'assurer que le S.I. proposé peut en théorie être réalisé avec les ressources disponibles. Le mot réalisable doit toutefois être pris dans un sens restreint, car l'outil de simulation ne permet pas d'évaluer la difficulté d'implémentation du système.

3. UNE METHODE DE CONSTRUCTION DE PROGRAMMES : LA PROGRAMMATION STRUCTUREE DE JACKSON (PSJ)

3.1. Présentation précise

3.1.1. Types de programme considérés

La méthode PSJ est une méthode de construction de programme qui s'applique à un certain type de programme. Il s'agit de programmes recevant en entrée un flux de données séquentiel et produisant en sortie un autre flux de données séquentiel. La méthode peut être adaptée au cas de programmes plus généraux présentant, par exemple, plusieurs flux de données en entrée ou en sortie ; nous nous limiterons dans cette présentation au cas le plus simple.

3.1.2. Représentation des programmes

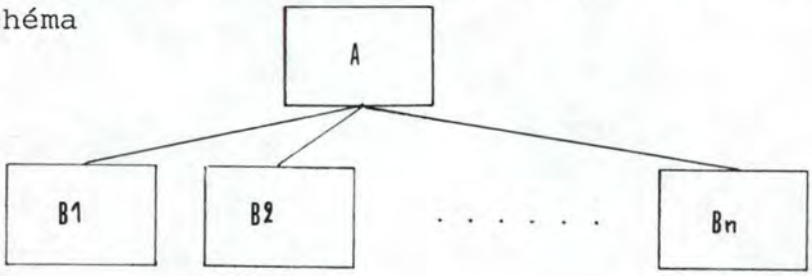
Dans cette méthode, un programme sera représenté par un arbre constitué à partir des quatre types de composants suivants :

a. Composant-séquence

Un composant séquence représente un morceau de programme dont l'exécution correspond à l'exécution successive d'un ou plusieurs autres morceaux de programme. Ces morceaux de programme seront représentés eux-mêmes par des composants que nous appellerons sous-composants-séquence du composant considéré.

Un composant-séquence est schématisé par un rectangle à l'intérieur duquel est inscrit son nom et en dessous duquel on schématisera les sous-composants.

Le schéma

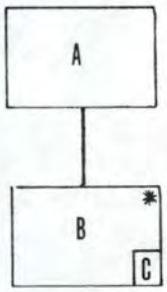


signifie que l'exécution du composant A est constituée des exécutions successives des composants B1, B2,... Bn (dans cet ordre).

b. Composant-itération

Un composant-itération représente un morceau de programme dont l'exécution correspond à O, une, ou plusieurs exécutions d'un même morceau de programme. Ce morceau de programme sera représenté par un composant que nous appellerons sous-composant-itération. Un composant-itération est schématisé par un rectangle à l'intérieur duquel on écrira son nom et en dessous duquel on schématisera le sous-composant par un rectangle comprenant son nom, un astérisque et une condition d'itération.

Le schéma :



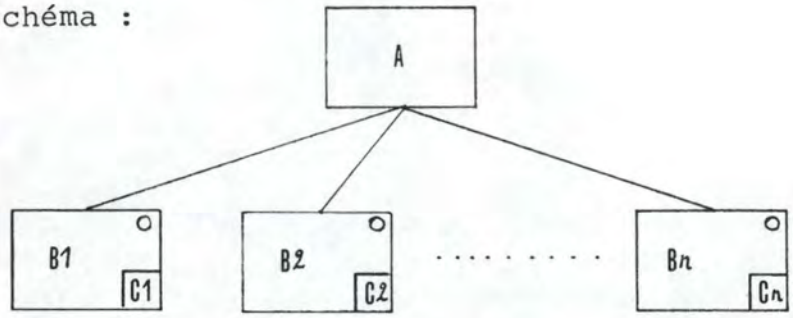
signifie : l'exécution de A consiste à exécuter B tant que la condition C est vraie.

c. Composant-sélection

Un composant-sélection représente un morceau de programme dont l'exécution correspond à celle d'un morceau de programme choisi parmi plusieurs selon la valeur d'une condition.

Les différents morceaux de programme pouvant être choisis seront représentés par un rectangle appelé sous-composant sélection du composant considéré. Chacun de ces rectangles contiendra le nom du sous-composant, le signe "O" et la condition d'exécution du sous-composant.

Le schéma :



signifie que l'exécution de A consiste à exécuter le composant B_i tel que C_i est vrai.
(Remarque : ceci suppose qu'une et une seule des conditions C_i soit vraie).

d. Action primitive

Un composant action primitive représente un morceau de programme "suffisamment simple" pour qu'on ne juge pas utile de le décomposer en parties plus élémentaires. Il est schématisé par une feuille de l'arbre représentant le programme.

Remarque : chaque composant constituant l'arbre d'un programme est sous-composant d'un et un seul autre composant, exception faite de la racine de l'arbre qui représente le programme tout entier.

3.1.3. Structuration des données

Une des étapes de la méthode PSJ consiste à définir la structure des données en entrée et en sortie. C'est sur base de ces structures que l'on déduira celle du programme. Le problème inhérent à cette étape est qu'il existe plusieurs manières de structurer les données. Toutes ces façons ne sont pas équivalentes dans la mesure où l'une peut amener à construire un programme relativement complexe et incompréhensible tandis que l'autre conduira à un programme nettement plus simple et compréhensible.

Parmi les différentes structurations de données pouvant se présenter on cherchera à découvrir celle qui "reflète le mieux la structure du problème à résoudre". Nous illustrerons par un exemple ce type de difficulté après avoir présenté le symbolisme de structuration des données.

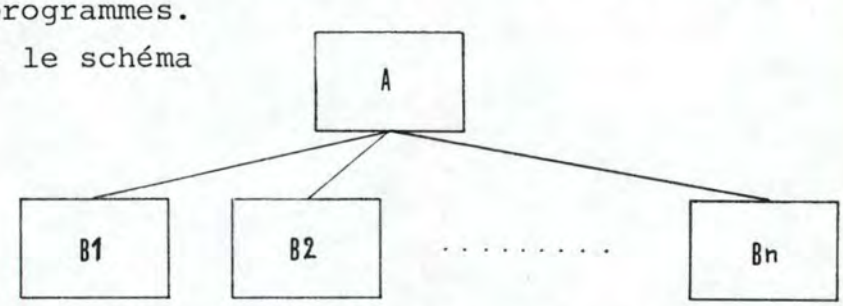
3.1.3.1. Types de composants

Un flux de données sera représenté (tout comme le programme) par un arbre. Cet arbre est constitué de composants d'un des quatre types mentionnés ci-dessous.

a. Composant-séquence

Un composant-séquence représente une classe de flux de données obtenus par concaténation de flux de données appartenant à plusieurs autres classes. Le symbolisme utilisé est le même que dans le cas des programmes.

Donc, le schéma



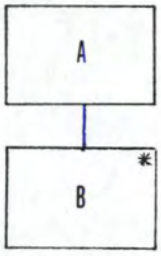
signifie qu'un flux de données de la classe A est formé par concaténation de n flux de données appartenant aux classes B1, B2,...Bn respectivement.

b. Composant-itération

Un composant-itération représente une classe de flux de données obtenus par concaténation de 0, 1, ou plusieurs flux de données appartenant à une même classe.

Le symbolisme utilisé est le même que dans le cas des programmes.

Donc, le schéma

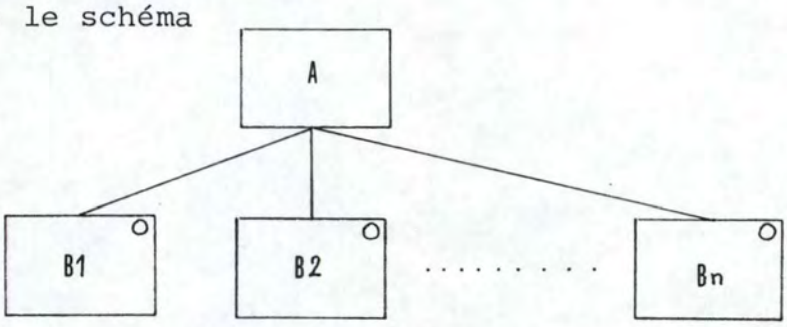


signifie : un flux de données de la classe A est formé par concaténation de 0, un ou plusieurs flux de données appartenant à la classe B.

c. Composant-sélection

Un composant-sélection représente une classe de flux de données égale à la réunion d'un ensemble d'autres classes de flux de données. Le symbolisme utilisé est le même que dans le cas des programmes.

Donc, le schéma



signifie : un flux de données de la classe A est un flux de données appartenant à une des classes B1, B2...Bn.

d. Composant élémentaire

Un composant élémentaire représente une classe de données élémentaires (flux réduits à un élément ou vides). Il est schématisé par une feuille de l'arbre représentant le flux de données.

3.1.3.2. Illustration des difficultés posées par la structuration des données

Pour illustrer la difficulté relative à la structuration des données, nous allons traiter un exemple.
Nous montrerons, sur cet exemple, que le flux de données en entrée peut être structuré de deux façons différentes (au moins).
Dans le paragraphe 3.1.7, nous insisterons sur le fait qu'une des deux structurations de données amènera à construire un programme nettement plus simple et compréhensible que l'autre.

Enoncé de l'exemple (Graas 85)

"Le département magasin d'une fabrique s'occupe des sorties et des entrées d'articles. Chaque entrée et chaque sortie est enregistrée sur une carte perforée. Cette carte contient le numéro de l'article, le code opération ("S" pour sortie, "E" pour entrée)et la quantité.
Les cartes sont déjà enregistrées dans un fichier et classées dans l'ordre croissant des numéros des articles. Ce fichier est dénommé : FSMT (Fichier Stock Magasin Trié).
Le programme à concevoir doit produire une liste résumée qui donne le solde après opération sur chaque article. Cet aperçu se présente comme suit :

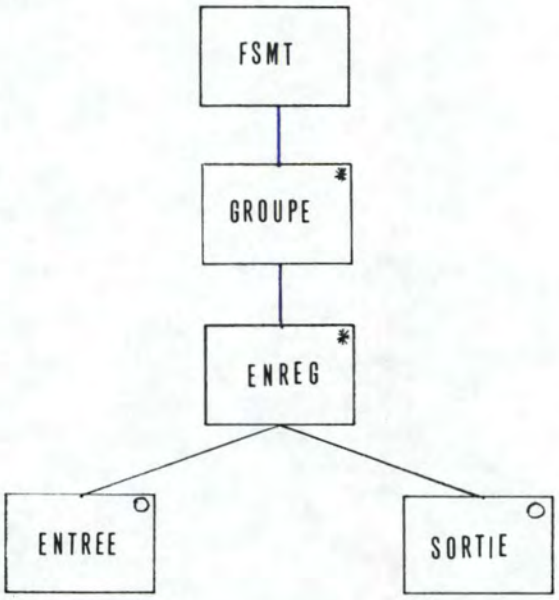
```
APERCU STOCK MAGASIN

A 1736    SOLDE OPERATION      - 450
A 1932    SOLDE OPERATION      +  35
...
...
...
Y 4640    SOLDE OPERATION    + 1.843

     FIN DU RAPPORT
```

."

Première possibilité de structuration du fichier

FSMT

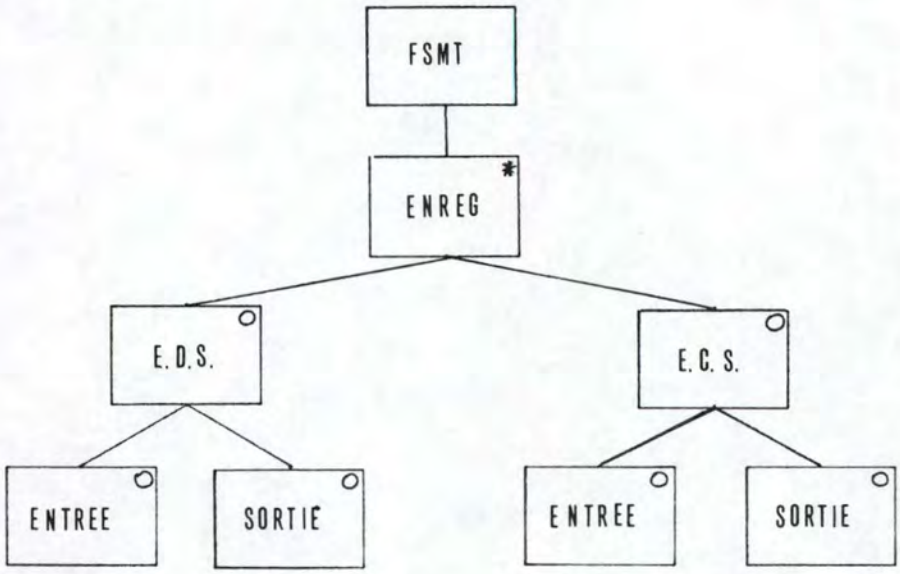
Le fichier FSMT est vu comme une suite de groupes. Un groupe est une suite d'enregistrements (ENREG.) de même numéro d'article.
Un enregistrement représente soit une entrée d'article, soit une sortie d'article. Un enregistrement entrée (ENTREE) contient le numéro de l'article, un code opération égal à "E" et une quantité. Un enregistrement sortie (SORTIE) contient le numéro de l'article, un code opération égal à "S" et une quantité.



Remarque : le formalisme proposé ne permet de décrire qu'incomplètement la structure du fichier : il ne permet pas d'exprimer que celui-ci est trié ni que tous les enregistrements d'un groupe ont même numéro d'article.

Deuxième possibilité de structuration du fichier FSMT

Le fichier FSMT est vu comme une suite d'enregistrements. Un enregistrement est soit un enregistrement de début de séquence, (E.D.S.), soit un enregistrement de continuation de séquence (E.C.S.). Un enregistrement de début (continuation) de séquence est un enregistrement qui n'est pas précédé (est précédé) d'un enregistrement de même numéro d'article. Les enregistrements de début ou de continuation de séquence sont soit des enregistrements entrée, soit des enregistrements sortie.
Un enregistrement entrée (sortie) contient le numéro de l'article, un code opération égal à "E" ("S") et une quantité.



On admet, cf (Jackson 75), que cette deuxième façon de structurer le fichier FSMT est beaucoup moins claire (reflète moins la structure du problème) que la précédente. On verra par la suite qu'il est difficile de construire un programme en se basant sur cette deuxième structuration.

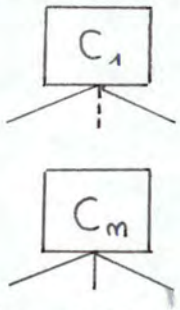
On peut déjà tenter de justifier que la première est plus simple en remarquant que dans ce cas tout flux constitutif peut être caractérisé à partir de ses composants (par exemple : un groupe est un ensemble d'enregistrements de même numéro) alors que dans le second cas, les composants E.D.S. et E.C.S. sont caractérisés à partir des composants qui les précèdent dans le fichier.
Dans le premier cas, la définition d'un composant se suffit à elle-même (a un sens indépendant du problème général posé) pas dans le second.

3.1.4. Définition de la notion de correspondance

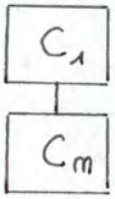
Une des étapes dans la construction d'un programme selon la méthode de Jackson consiste à rechercher une correspondance entre les structures des entrées et des sorties. Nous expliquerons ci-dessous ce qu'il faut entendre par cette notion de correspondance. Au préalable, nous définirons un certain nombre de concepts nécessaires à la définition du terme "correspondance".

- Le concept d'arbre devra être pris dans le sens d'arbre construit à partir des quatre types de composants vus précédemment.

- Un arbre A est un <u>élagage</u> d'un arbre B ssi A peut être obtenu
  1) en supprimant des feuilles ou des sous-arbres de B,
  2) en remplaçant certains chemins de B .



par des arcs :



- Un arbre A peut être mis en correspondance formelle avec un arbre B ssi (par définition) il existe un arbre C tel que A et B sont tous deux des élagages de C.
  On peut choisir C de telle sorte que tout composant de C se retrouve au moins dans A ou dans B. On choisira toujours C de cette sorte en pratique (On l'appellera fusion de A et B, selon la correspondance choisie).
  Dans une telle correspondance formelle, on dira qu'un composant de A est en correspondance avec un composant de B s'ils sont tous deux obtenus par élagage à partir du même composant de C.

- Soient A l'arbre représentant la structure des entrées d'un programme et B l'arbre représentant celle des sorties.
  On appellera CORRESPONDANCE entre A et B, une correspondance formelle entre A et B vérifiant les conditions suivantes :
  1) pour tout couple C_A, C_B de composants de A et B en correspondance, le flux d'entrée contient toujours autant d'occurrences de C_A que le flux de sortie en contient de C_B.
  2) si n est ce nombre d'occurrences, $\forall i : 1 \leqslant i \leqslant n$, la ième occurrence de C_B peut être générée (obtenue) à partir de la ième occurrence de C_A.

3.1.5. Construction du programme

Dans la méthode de Jackson, la conception d'un programme se déroule en 4 étapes :

1. On spécifie le problème à traiter.
2. On donne une description des données en entrée et en sortie ; on définit la structure des entrées et des sorties par 2 arbres (soient A et B respectivement).

3. On recherche une correspondance entre A et B comme indiqué au paragraphe 3.1.4. Soit C la fusion de A et B pour cette correspondance.

4. On complète l'arbre C du programme
   - en ajoutant des conditions aux sous-composants itération et aux sous-composants condition de C ;
   - en ajoutant des composants actions primitives. (*)

La définition de cette 4ème étape est particulièrement peu précise. En fait, cette imprécision reflète la difficulté qu'il y a à donner des règles pour déterminer les conditions et actions-primitives à ajouter à la structure du programme.
Nous donnerons au paragraphe 3.1.6. quelques ébauches de telles règles ; malheureusement elles restent encore bien trop vagues.

3.1.6. Détermination des conditions et actions primitives (tentative)

La détermination des conditions et actions primitives se fait en se basant sur le type des composants de C et sur les correspondances entre les composants de A et de B.

Tout composant de C est (représente) un morceau de programme qui traite en entrée une partie (peut-être vide) du flux d'entrée et génère en sortie une partie (peut-être vide) du flux de sortie. Ceci peut être précisé en se basant sur les correspondances.
Soit C_o le composant considéré de C. Trois cas principaux sont à considérer :

1) C_o correspond à deux composants de A et B mis en correspondance.
2) C_o correspond à un composant de A mais pas de B.
3) C_o correspond à un composant de B mais pas de A.

(*) Rem. : L'ajout des actions primitives entraine parfois la création de niveaux supplémentaires dans l'arbre (Composants séquences).

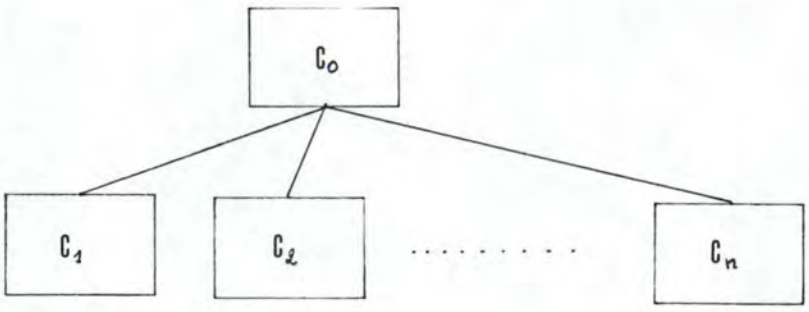
Remarque :

Dans le premier cas, une exécution de C_o, traitera effectivement une partie du flux d'entrée et générera la partie correspondante du flux de sortie. C'est le cas "normal".
Dans le deuxième cas, il y a lecture sans génération. Cela signifie (en général) qu'une partie de l'information contenue dans le flux d'entrée sera mémorisée pour être générée plus tard (cas particulier : partie inutile du flux d'entrée).
Dans le troisème cas, il y a génération sans lecture, c'est le cas inverse du précédent : une partie du flux de sortie est généré, non à partir du flux d'entrée mais d'une information mémorisée précédemment. (fin de la remarque).

Pour justifier l'applicabilité de la méthode, il faudrait étudier tous les cas pour montrer chaque fois qu'on peut construire le programme en complétant l'arbre C par des conditions et des actions primitives. Nous envisagerons seulement le premier cas pour simplifier. Remarquons cependant que l'abondance des cas possibles montre que la méthode ne peut être appliquée sans une bonne dose d'imagination.

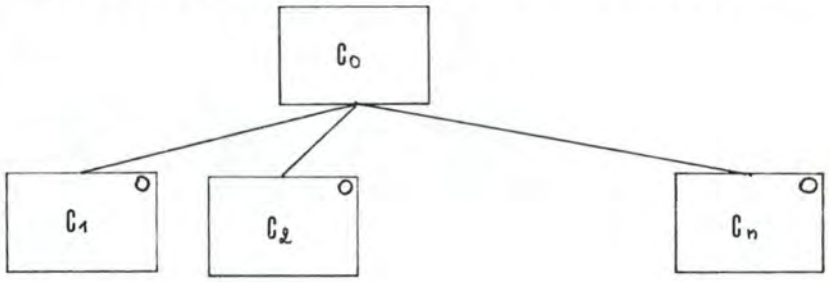
Construction d'un composant de programme correspondant à un composant du flux d'entrée et à un composant du flux de sortie.

Cas 1 : le composant est un composant séquence



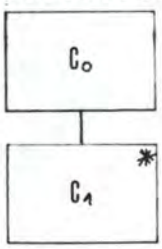
On suppose que C_o, C_1, $C_2 \ldots C_n$ correspondent à des composants à la fois des flux d'entrée et de sortie. Alors, une occurrence de C_o dans le flux d'entrée se compose d'une occurrence de C_1, suivie d'une occurrence de C_2, etc. L'exécution des composants C_1, suivie de celle de C_2, etc, générera l'occurrence correspondante de C_1 (dans le flux de sortie), suivie de celle de C_2, etc. Le tout formera bien l'occurence de C_o dans le flux de sortie correspondant à celle de C_o dans le flux d'entrée. Ceci démontre que le schéma ci-dessus est correct par rapport à sa spécification.

Cas 2 : Le composant est un composant condition



Il suffit d'ajouter aux composants $C_1 \ldots C_n$ une condition permettant de savoir si le flux d'entrée appartient à la classe C_1 ou à la classe $C_2, \ldots,$ ou à la classe C_n. La manière dont ces conditions seront réalisées dépend de la nature du flux d'entrée.

Cas 3 : Le composant est un composant itération



Une occurrence du flux d'entrée de la classe C_o est formée de n occurrences de la classe C_1. Il suffit d'exécuter n fois le composant C_1 pour générer les n occurrences des flux de sortie de la classe C_1 composant le flux de sortie de la classe C_o. Pour cela on cherchera une condition permettant de "détecter" la fin de la dernière occurrence du flux d'entrée de la classe C_1.

Cas 4 : Le composant est un composant "élémentaire"

Une exécution du composant C_o doit traiter un élément du flux d'entrée et générer un élément du flux de sortie. Le composant C_o contiendra un ensemble d'actions primitives permettant de réaliser ce traitement.

Conclusion

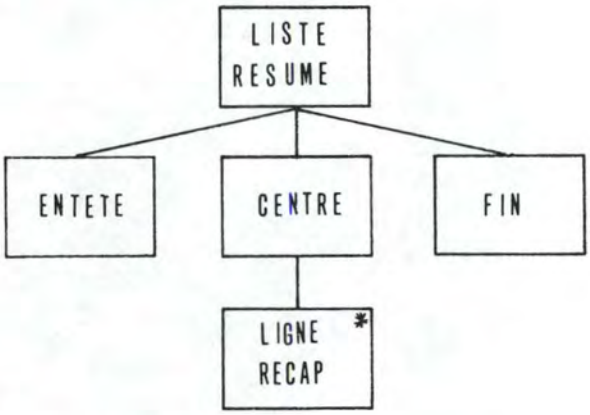
La "méthode" pour déterminer les actions et les conditions vue ci-dessus, est trop vague pour pouvoir être qualifiée de systématique. Il est difficile d'être plus précis parce que selon le problème traité, les flux d'entrée et de sortie peuvent prendre des formes très différentes.
La méthode est "relativement" systématique jusqu'à la constitution de l'arbre fusionné des structures d'entrée et de sortie (étape 3 dans la construction du programme pt 3.1.5.).
Après cela, si l'on veut travailler proprement, il faut, en fonction du problème, donner des spécifications précises aux différents composants de l'arbre du programme permettant alors de construire de manière exacte ces composants, selon le principe de la méthode descendante.
L'intérêt principal de la méthode de Jackson sera donc de suggérer une "bonne" découpe du problème basée sur la correspondance entre les entrées et les sorties.

3.1.7. Illustration

Nous avons expliqué au paragraphe 3.1.5. les différentes étapes à suivre pour construire un programme.
Nous voudrions ci-dessous montrer que le choix de la structure des entrées et des sorties (étape 2) détermine la qualité du programme à construire.

Nous illustrerons ce fait en reprenant le problème énoncé au paragraphe 3.1.3.2. Nous y avions donné 2 possibilités de structuration des entrées. Après avoir défini la structure des sorties, nous allons chercher à découvrir une correspondance entre chaque structure des entrées et la structure des sorties.

3.1.7.1. Structuration des sorties



La liste "résumé" est vue comme la succession d'une ligne en tête de rapport suivie d'un ensemble de lignes récapitulatives suivi d'une ligne fin de rapport.

Désignons par :

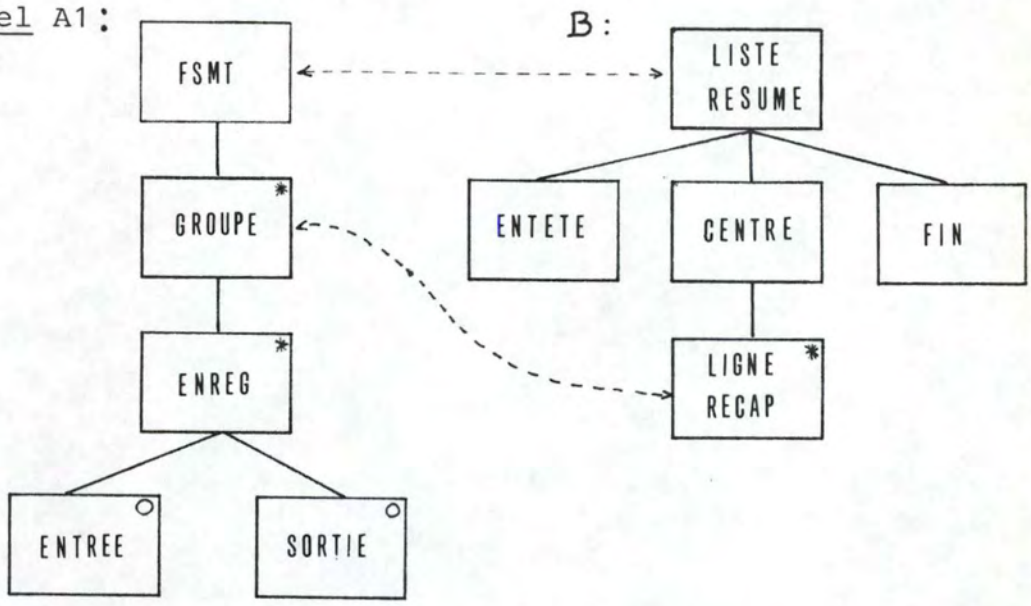
B l'arbre représentant la structure des sorties

A1 l'arbre représentant la première possibilité de structuration des entrées

A2 l'arbre représentant la deuxième possibilité de structuration des entrées

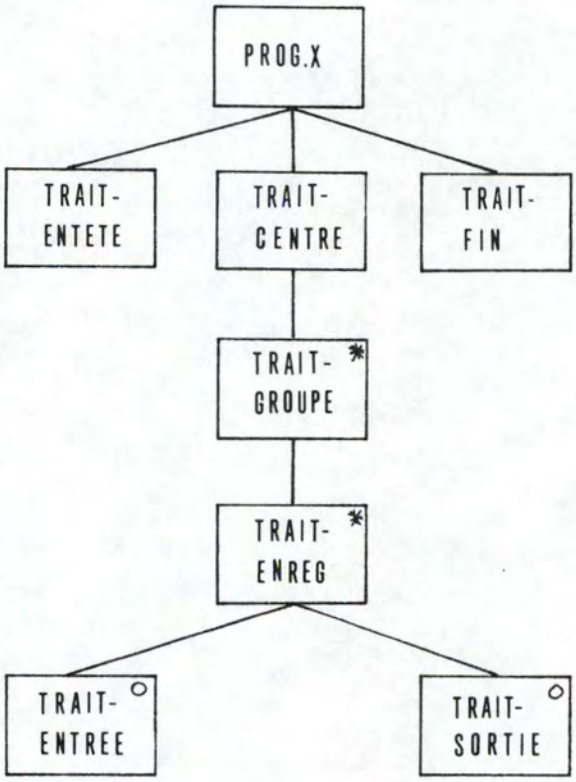
3.1.7.2. Recherche d'une correspondance entre A1 et B

Rappel A1 :



En essayant de mettre les 2 arbres A1 et B en correspondance nous obtiendrons un arbre fusion de A1 et B (désigné par C).

C :



En élaborant cette structure intermédiaire C on a bien mis A1 en correspondance avec B.
En effet :

- l'arbre A1 est un élagage de C puisqu'il peut être obtenu à partir de C en supprimant les feuilles TRAIT - ENTETE, TRAIT - FIN et en remplaçant le chemin partant du composant PROG. X et aboutissant au composant TRAIT. GROUPE par un arc.
- l'arbre B est un élagage de C puisqu'il peut être obtenu à partir de C en supprimant les composants TRAIT-ENREG, TRAIT-ENTREE, TRAIT-SORTIE

d'où on en déduit que A peut-être mis en correspondance formelle avec B. Dans cette correspondance formelle les composants GROUPE, LIGNE RECAP sont en correspondance, ainsi que les composants FSMT, LISTE RESUME.
De plus :

- pour les couples (FSMT, LISTE RESUME), (GROUPE, LIGNE RECAP) de composants de A1 et B en correspondance, il y a toujours autant d'occurrences du composant FSMT (GROUPE) que de LISTE RESUME (LIGNE RECAP)

et si n est ce nombre d'occurrences, $\forall$ i $1 \leqslant i \leqslant n$, la ième occurrence de LISTE RESUME (LIGNE RECAP) peut être générée à partir de la ième occurrence de FSMT (GROUPE).

Il existe donc bien une correspondance entre A1 et B selon la définition donnée au paragraphe 3.1.4.

L'arbre C représente la structure du programme à construire pour résoudre le problème donné. Il reste à ajouter les conditions et actions primitives. Mais avant cela, il est nécessaire d'attacher à chaque composant de l'arbre C une spécification. Cette activité est facilitée, dans le cas présent, par la simplicité des règles de correspondance. Nous le montrerons, ci-dessous, en spécifiant les composants principaux de l'arbre C à savoir TRAIT-CENTRE, TRAIT-GROUPE, TRAIT-ENREG.

Définitions

1. Un groupe est un ensemble d'enregistrements de même numéro d'article, chacun de ces enregistrements représente une opération effectuée sur cet article (opération d'entrée ou de sortie).
2. Une ligne récapitulative est constituée d'un numéro d'article, de la suite de caractères "solde opération" et du montant du solde opération de cet article.
3. Le solde opération d'un article de numéro x est égal à la somme des montants des opérations d'entrée effectuées sur l'article de n° x diminuée de la somme des montants des opérations de sortie effectuées sur le même article.

Spécification de TRAIT-CENTRE

- lit n ($\geqslant$ 0) groupes et génère n lignes récapitulatives ;
- le permier enregistrement du premier groupe a déjà été lu s'il existait sinon la fin de fichier a été déclenchée (précondition).

Spécification de TRAIT-GROUPE

- lit un groupe dont le premier enregistrement a déjà été lu ;
- génère la ligne récapitulative correspondant à ce groupe ;
- lit le premier enregistrement suivant ce groupe, s'il existe, sinon déclenche la fin de fichier.

Spécification de TRAIT-ENREG.

- reçoit un enregistrement représentant une opération effectuée sur un article de numéro égal à NOART ;
- ajoute le montant de cette opération au contenu de la variable SOLDE si c'est une opération entrée, sinon la retire ;

- lit l'enregistrement suivant s'il existe, sinon déclenche la fin de fichier.

3.1.7.3. Recherche d'une correspondance entre A2 et B

(Une compréhension complète de ce paragraphe nécessite la lecture préalable du paragraphe 3.2.).

Il est impossible de trouver une correspondance entre A2 et B. On dit qu'il y a conflit de structure. On peut essayer de résoudre cette situation en appliquant les solutions habituellement proposées (voir paragraphe 3.2.3.). Mais en suivant cette voie, on se rend compte que la solution reviendrait à créer un flux intermédiaire qui aurait à la fois la "bonne" structure A1 et la "mauvaise" structure A2. Dans ce cas, le programme produisant le flux intermédiaire à partir du flux d'entrée n'aurait d'autre but que de restituer le flux d'entrée tel quel. La solution consiste donc à remplacer la mauvaise structure par la bonne. Pour éliminer le conflit de structure entre A2 et B on pourrait procéder autrement : modifier légèrement les structures en entrée et en sortie en s'arrangeant pour qu'il y ait correspondance. Pour ce faire, on peut partir d'une idée intuitive de l'exécution d'un programme dont la structure serait proche de A2. Chaque fois que l'on rencontre un enregistrement de début de séquence (E.D.S.) il faut clôturer le groupe précédent en générant la ligne récapitulative lui correspondant et initialiser le nouveau groupe.

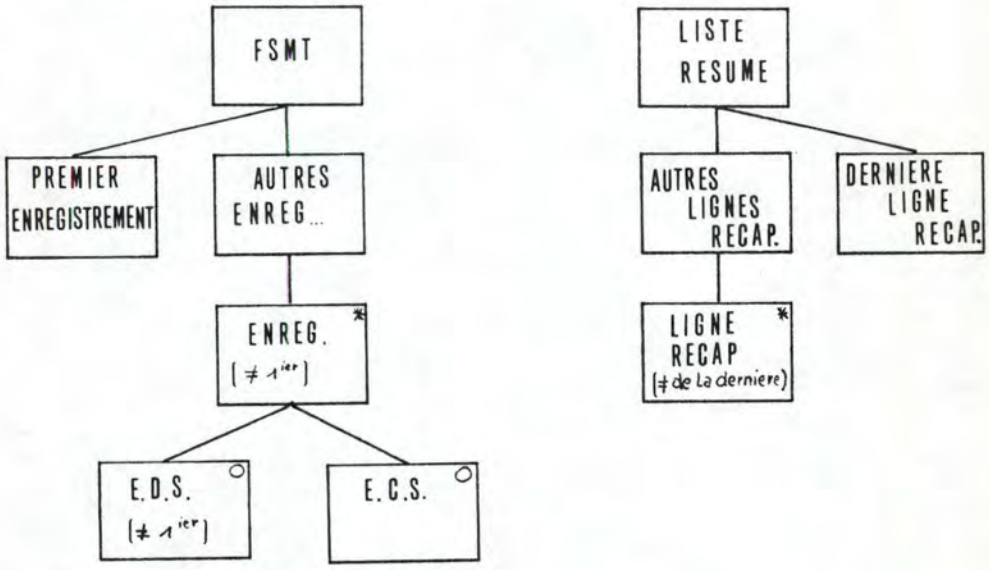
Ce traitement ne peut cependant pas être réalisé dans son entièreté lorsqu'on rencontre le premier enregistrement du fichier. En effet, dans ce cas la notion de groupe précédent n'a pas de sens, il est donc nécessaire de créer un traitement spécial pour le premier enregistrement. Ces 2 types de traitement ne suffisent pas, car si on s'en

contente, la clôture du dernier groupe ne sera jamais réalisée, on créera donc un traitement spécial pour le dernier enregistrement du fichier de sortie.

En se basant sur ces idées :

- le fichier d'entrée est vu comme une suite constituée du premier enregistrement suivi des autres enregistrements ;
- le fichier de sortie est vu comme une suite constituée de l'ensemble des enregistrements (à l'exception du dernier) suivie de ce dernier enregistrements.

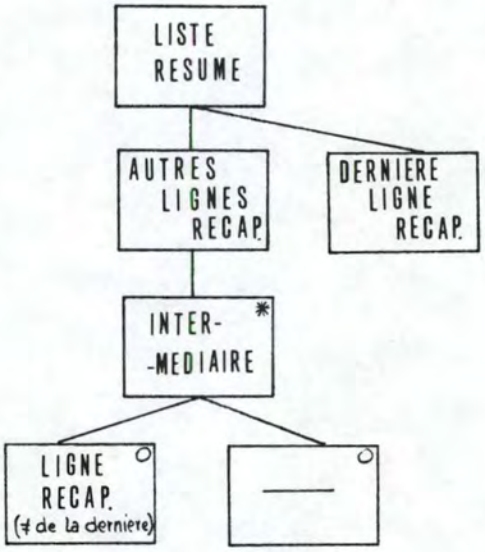
On en déduit les 2 arbres ci-dessous :



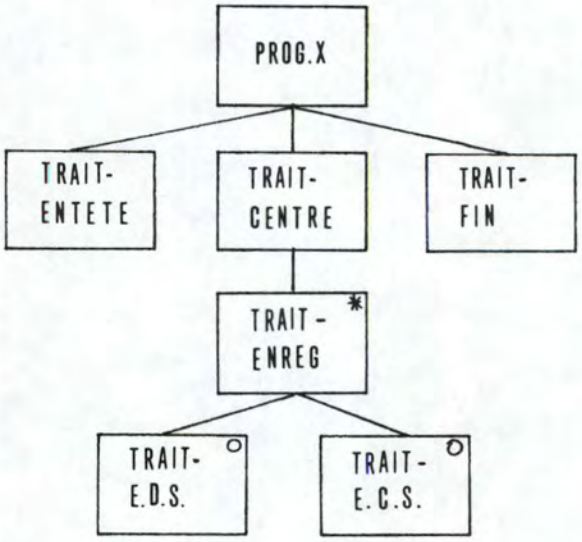
(La définition des flux correspondant aux différents composants se déduit de la discussion précédente).

Ces deux arbres ne peuvent pas encore être mis en correspondance parce qu'il n'y a pas autant d'occurrences de ligne récapitulative (différente de la dernière) que d'enregistrements ENREG (différents du premier) dans le fichier d'entrée. Pour résoudre ce dernier problème il suffit de remplacer le composant ligne récapitulative (différente de la dernière) par un composant

condition "fictif" intermédiaire ayant autant d'occurrences que le nombre d'enregistrements du fichier d'entrée, moins un et admettant deux sous-composants dont les occurrences sont soit des lignes récapitulatives soit "vides". On obtient finalement l'arbre sortie suivant qui peut être mis de manière évidente en correspondance avec l'arbre d'entrée.



En fusionnant les deux arbres on "obtient" la structure suivante pour le programme



Il serait exagéré de dire que cette structure a vraiment été obtenue en appliquant la méthode de Jackson car elle correspond exactement à l'idée intuitive de la page 64
C'est donc du programme qu'on a déduit la structure des fichiers et non l'inverse.

Pour illustrer le fait que cette manière de procéder conduit à un programme plus compliqué et en tout cas moins clair, nous donnons ci-dessous les spécifications des différents composants du programme.

Spécification de TRAIT-ENTETE

(1) Ouvre les fichiers d'entrée et de sortie ;
(2) génère un enregistrement d'en-tête dans le fichier de sortie ;
(3) lit le permier enregistrement du fichier d'entrée et initialise - une variable NOART avec son n° article
  - une variable SOLDE avec son montant (si c'est une entrée), l'opposé de son montant (si c'est une sortie) ;
(4) lit le deuxième enregistrement s'il existe (sinon déclenche la fin de fichier).

Spécification de TRAIT-CENTRE

(1) Lit les autres enregistrements du fichier d'entrée (jusqu'à déclencher la fin de fichier) ;
(2) écrit sur le fichier sortie l'ensemble des lignes récapitulatives sauf la dernière ;
(3) se termine en ayant placé dans NOART et SOLDE le n° d'article et le solde de la dernière ligne récapitulative.

Spécification TRAIT-ENREG.

(1) Si l'enregistrement courant (dernier lu) à n° article égal à NOART, son montant est ajouté à la variable SOLDE (si c'est une entrée) et retiré sinon.
Sinon, une ligne récapitulative est écrite sur le fichier de sortie avec le n° d'article NOART et le solde SOLDE ; ensuite le montant de l'enregistrement courant est placé dans SOLDE (si c'est une entrée) ou l'opposé de son montant (si c'est une sortie), le numéro d'article de l'enregistrement courant est placé dans NOART.

(2) On lit l'enregistrement suivant du fichier d'entrée s'il existe (sinon la fin de fichier est déclenchée).

Spécification TRAIT-FIN

(1) Ecrit une ligne récapitulative avec le numéro d'article NOART et le solde SOLDE.

(2) Ecrit une ligne de fin de rapport.

(3) Ferme les fichiers.

3.2. Limites et extensions possibles

3.2.1. Cas de plusieurs flux d'entrée et de sortie

Il est possible d'étendre la méthode au cas de plusieurs flux d'entrée et de sortie. Nous ne l'expliquerons pas en détail ici.
Intuitivement, on peut le justifier comme suit : il est toujours possible de construire une structure de données hypothétique correspondant à une fusion des différentes entrées ou des différentes sorties reflétant l'ordre dans lequel elles sont traitées ou générées.

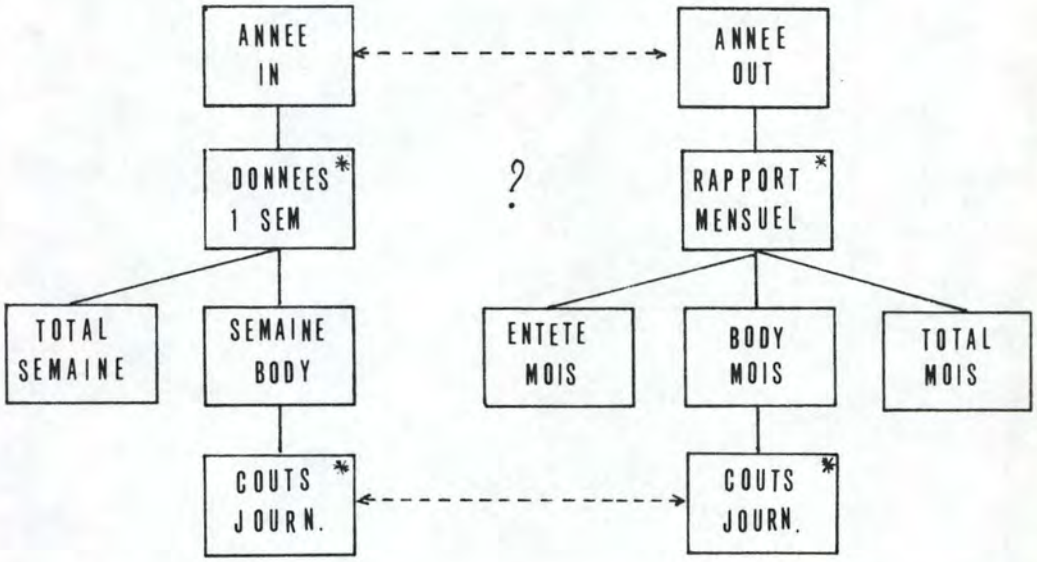
3.2.2. Les conflits de structure

Supposons qu'on ait défini pour un problème donné, un arbre A représentant une "bonne" structure des entrées et un arbre B représentant une "bonne" structure des sorties.
On dit alors qu'il y a conflit de structure entre A et B si on ne peut pas établir une correspondance entre A et B selon les règles définies plus haut.
On distingue plusieurs formes de conflits de structure (classification non exhaustive) :

- conflit d'ordre : ce type de conflit apparait lorsque les composants des deux structures de données ne se présentent pas dans le même ordre (Graas, 85) ;
- conflit d'entrelacement : ce type de conflit se présente lorsque deux composants de A et B ne peuvent être mis en correspondance parce que les données qui constituent les occurrences de ces composants sont regroupées selon des critères différents ;
- conflit de frontière : ce type de conflit survient lorsque les 2 structures de données possèdent chacune 2 ou plusieurs niveaux d'itération et que
  1. les composants d'itération au niveau supérieur correspondent bien,

2. les parties itérées au niveau inférieur correspondent bien mais
3. il n'existe aucune correspondance entre les composants au niveau intermédiaire.

Exemple : Une firme désire sortir un rapport sur l'année écoulée, elle désire donner un aperçu des coûts journaliers pour chaque mois ainsi qu'un total mensuel des coûts.
A cet effet, la firme dispose d'un fichier groupant et totalisant par semaine les coûts journaliers.
Les structures de données, avec les correspondances indiquées, se présentent comme suit :



Il règne une parfaite correspondance au niveau le plus haut et au niveau le plus bas. Au niveau intermédiaire il y a conflit (on ne peut pas dire que le composant rapport mensuel est une itération de rapport hebdomadaire car un mois ne comprend pas un nombre entier de semaines).

3.2.3. Solutions aux conflits de structure

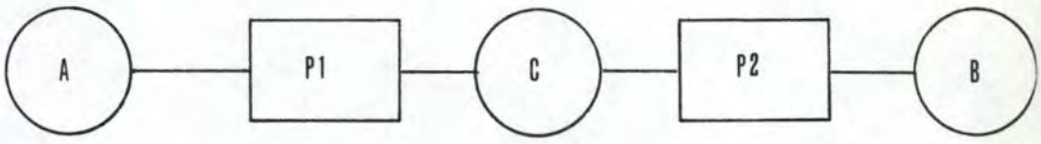
Les conflits d'ordre sont résolus par l'utilisation d'un programme de tri ou une table ou fichier à accès direct.

Les solutions aux conflits de frontière et d'entrelacement sont moins évidentes. Elles sont semblables dans la mesure où la solution au conflit d'entrelacement est une généralisation de la solution apportée au conflit de frontière (création de plus d'un fichier intermédiaire).
Nous expliquerons ci-dessous comment le conflit de frontière peut être résolu. Il s'agit tout d'abord de définir un flux intermédiaire entre le flux des entrées (représenté par l'arbre A) et le flux des sorties (représenté par l'arbre B).Ce flux intermédiaire sera réalisé sous forme d'un fichier appelé fichier intermédiaire. Sur ce flux intermédiaire, on doit pouvoir placer deux structures différentes (représentées par les arbres) C et D telles que les structures A, C (resp D, B) soient en correspondance. Ensuite, il ne restera plus qu'à construire deux programmes P1, P2 en se basant sur chacune des correspondances.
Cette solution qui consiste à construire 2 programmes simples reliés entre eux à l'aide d'un FICHIER séquentiel n'est cependant pas très efficace.
En effet, les données du flux intermédiaire ne peuvent être exploitées (par le programme P2) que lorsque le programme P1 a produit l'entièreté du FICHIER intermédiaire.
On voudrait améliorer cette solution, en supprimant le fichier intermédiaire et en rendant accessible au programme P2 une donnée du flux intermédiaire dès qu'elle est produite.
Ainsi on voudrait pouvoir exécuter P1 jusqu'au moment où il aura produit un enregistrement du flux de données intermédiaires, ensuite l'ajourner, exécuter P2 jusqu'au moment où il aura besoin de l'élément suivant du flux intermédiaire, puis ajourner P2 et remettre P1 en marche, etc...
Ceci est possible en appliquant aux programmes P1 et P2 des modifications, utilisant une technique appelée <u>inversion</u>.

3.2.4. Inversion de programme

3.2.4.1. Principes

Supposons que pour résoudre le conflit de structure entre les flux A et B, nous ayons été amenés à construire deux programmes P1, P2 reliés par un flux de données intermédiaire.



Nous pouvons améliorer cette solution en supprimant le fichier FICHIER intermédiaire C :

- soit en inversant P1 par rapport à la sortie C, dans ce cas P1 devient un sous-programme de P2 : au lieu de lire directement un enregistrement de C, P2 appellera le programme P1 qui lui fournira cet enregistrement (amélioration 1) ;
- soit en inversant P2 par rapport à l'entrée C, dans ce cas P2 devient un sous-programme de P1 : au lieu d'écrire un enregistrement de C, P1 appellera le programme P2 auquel il transfèrera cet enregistrement (amélioration 2).

Nous allons montrer et justifier les modifications qu'il faut apporter à P1 et P2 si on opte pour l'amélioration 1 (la démonstration se fait de manière plus ou moins analogue pour l'amélioration 2, il ne nous a pas semblé nécessaire de la reprendre ici).

3.2.4.2. Règles d'inversion et justification

Pour justifier les règles d'inversion appliquées à P1 et P2 il faut donner une "spécification" précise d'un appel de P1 inversé (effectué par P2).

Supposons que l'exécution de P1 non inversé produise un fichier intermédiaire C comportant N enregistrements.

Alors le sous-programme P1 inversé ($P1^{-1}$) sera appelé N + 1 fois par P2 et le ième appel de $P1^{-1}$ aura l'effet suivant :

- pour i tel que $1 \leqslant i \leqslant N$: un indicateur EOF est renvoyé avec la valeur "O" ; le ième enregistrement de C est fourni au programme P2 ;
- pour i = N + 1 : l'indicateur EOF est renvoyé avec la valeur "1".

a. Modifications apportées à P2 (donnant $P2^M$)

On suppose que P2 contient :

- une instruction OPEN INPUT C,
- des instructions MOVE "O" TO EOF,
  READ C INTO REC AT END MOVE "1" TO EOF,
- une instruction CLOSE C.

P2 est modifié comme suit :

- les instructions OPEN INPUT C CLOSE C sont supprimées,
- les autres groupes d'instructions sont remplacés par CALL P1MOINS1 USING EOF, REC.

b. Justification

Montrons que $\forall i : 1 \leqslant i \leqslant N+1$, le préfixe de l'exécution de $P2^M$ se terminant avant le ième appel de $P1^{-1}$ a même effet sur les variables en working-storage-section (W.S.S.) et sur le flux de sortie que l'exécution de P2 se terminant avant la ième exécution de l'instruction READ C.

En effet, c'est vrai pour i = 1 car les exécutions des 2 programmes sont identiques (à l'OPEN OUTPUT C près) jusqu'au premier appel de $P1^{-1}$ (correspondant au premier READ).

Supposons que ce soit vrai pour $i < N+1$.
La ième exécution du READ C va placer dans REC le ième enregistrement de C et donner à EOF la valeur O. De même le ième appel de $P1^{-1}$ (selon la spécification d'un appel de P1 inversé).
Donc, le contenu de la W.S.S. et du flux de sortie sera encore le même pour P2 et $P2^M$. Par conséquent, les exécutions des 2 programmes se dérouleront encore

de la même façon jusqu'à la prochaine exécution de READ C (dans P2) et jusqu'au prochain appel de $P1^{-1}$ (dans $P2^{M}$).
On en déduit que l'exécution de P2 parviendra à la N+1 exécution de READ C, dans les mêmes conditions que l'exécution de $P2^{M}$ au N+1 appel de $P1^{-1}$.
Dans les 2 cas EOF recevra donc la valeur 1.
Par conséquent, les exécutions des 2 programmes se termineront de la même façon (sauf que le CLOSE C sera exécuté dans P2 mais cela n'a pas d'influence sur le contenu de la W.S.S. ni sur le flux de sortie).

C. Modifications apportées à P1 (donnant $P1^{-1}$)

On suppose que P1 contient :
- une instruction OPEN OUTPUT C,
- des instructions WRITE C FROM REC1
- une instruction CLOSE C,
- une instruction STOP RUN.

On construit $P1^{-1}$ comme suit :
- on introduit une LINKAGE SECTION contenant
  - la description COBOL de la zone REC destinée à recevoir les enregistrements successifs de C,
  - la description COBOL de l'indicateur EOF ;
- on ajoute en W.S.S. une variable entière QS (qui servira à repérer les différents points de reprise de l'exécution dans $P1^{-1}$) initialisée à 1 ;
- soit M le nombre d'instructions WRITE C figurant dans P1.

P1 est modifié comme suit :
- on fait précéder la première instruction de P1 de la séquence : ENTRY P1MOINS1 USING EOF REC.
  GO TO Q1,...., QM+1 USING QS.
  Q1.
- les instructions OPEN OUTPUT C et CLOSE C sont supprimées ;
- on numérote les instructions WRITE C de 2 à M+1,

on remplace la ième par la séquence :

```
MOVE i TO QS.
MOVE O TO EOF.
MOVE REC1 TO REC.
GOBACK.
Qi.
```

- on remplace l'instruction STOP RUN par la séquence :

```
MOVE 1 TO EOF.
GOBACK.
```

d. Justification

Soit, rappelons-le, N le nombre d'enregistrements du fichier C.
Il faut montrer que $P1^{-1}$ respecte sa "spécification" (voir page 72).
Pour cela, on montrera d'abord que $\forall$ i : $1 \leqslant i \leqslant N$ le ième appel de $P1^{-1}$

- renverra dans REC le ième enregistrement de C,
- renverra dans EOF la valeur O,
- donnera à QS la valeur du numéro de l'instruction WRITE C, ayant servi à créer ce ième enregistrement,
- laissera la W.S.S. et le flux d'entrée dans l'état qu'ils avaient au moment de l'exécution de cette instruction WRITE.

Démonstration

- pour i = 1

lors du premier appel de $P1^{-1}$, on aura QS = 1.
Par conséquent l'exécution de $P1^{-1}$ continuera à partir de Q1 qui coincide avec le début de P1.
Donc l'exécution de $P1^{-1}$ se déroulera comme celle de P1, depuis son début, jusqu'à l'exécution d'un ordre WRITE C (soit j le numéro de cet ordre).

Dans $P1^{-1}$ la séquence :

MOVE j TO QS.

MOVE O TO EOF.

MOVE REC1 TO REC.

GOBACK.

sera exécutée.

Elle donnera à REC la valeur du ième enregistrement de C (contenue dans REC1);

à EOF la valeur O ;

à QS la valeur j (numéro de l'instruction WRITE, puis la main sera rendue au programme appelant avec un contenu de W.S.S. et un état du flux d'entrée identique à ce qu'ils étaient dans P1 au moment de l'exécution de l'ordre WRITE C.

- vrai pour i => vrai pour i + 1.

Considérons le i + 1ème appel de $P1^{-1}$.
Par hypothèse de récurrence, le ième appel aura laissé le contenu de la W.S.S. et l'état du flux d'entrée comme ils étaient après exécution de l'instruction WRITE ayant généré le ième enregistrement de C. De plus, QS aura pour valeur le numéro j de cette instruction. Donc, l'instruction GO TO Q1,...., Q_{M+1} DEPENDING ON QS fera continuer l'exécution de $P1^{-1}$ au point Qj précédant exactement (dans $P1^{-1}$) l'instruction suivant l'instruction WRITE C (dans P1). Par conséquent, l'exécution de $P1^{-1}$ se continuera exactement comme celle de P1 après cette exécution de l'instruction WRITE C, jusqu'au moment où une nouvelle instruction WRITE C sera exécutée dans P1.

Le reste du raisonnement est le même que dans le cas i = 1.

Il faut encore justifier que $P1^{-1}$ vérifie sa spécification pour i = N+1.

D'après ce qui précède, le N+1 appel de $P1^{-1}$ provoquera une reprise de l'exécution de $P1^{-1}$, juste après la dernière instruction WRITE C exécutée dans P1, avec un état de la W.S.S. et du flux d'entrée identique. Donc l'exécution des deux programmes se dérouleront de la même façon jusqu'à l'exécution de STOP RUN dans P1. L'appel se terminera alors en renvoyant 1 dans EOF, après avoir exécuté les éventuelles opérations de clôture sur le flux d'entrée.

Remarques :

1. Avant d'effectuer toute modification aux programmes P1 et P2, il convient de remplacer les instructions PERFORM à l'aide des instructions conditionnelles IF.. et de branchements GO TO.... Ceci est dû à la manière dont le PERFORM est implémenté en COBOL.

2. Les règles d'inversion (et leur justification) ont été rédigées après les avoir appliquées sur un exemple concret dans la deuxième partie. Une simplification nous est apparue entretemps : le CLOSE C du programme P2 que nous avions remplacé par un N+2ème appel au sous programme $P1^{-1}$ peut être supprimé.

* * *

4. TESTS

La notion de test ayant déjà été présentée au paragraphe 4 du chapitre 1 (page 29) nous nous limiterons ici à énoncer quelques principes classiques de construction de test tirés de (Van Lamsweerde, 84).
Le principe de test se décompose en trois temps : dans un premier temps, on construit un jeu de données de test, ensuite on exécute le programme avec ce jeu, enfin on compare les résultats attendus avec ceux obtenus. Le problème des tests réside dans le fait qu'il est impossible d'effectuer un test pour chaque valeur des arguments en entrée d'un programme. La "solution" consiste alors à partitionner l'ensemble des valeurs possibles des arguments en entrée, en classes. Ensuite, pour chacune de ces classes, on choisit un "bon" représentant, c'est-à-dire une valeur telle que si le programme fournit des résultats corrects pour cette valeur (ce représentant), il y a de "fortes raisons" de croire qu'il fournira des résultats corrects pour les autres valeurs de cette même classe. Toute la difficulté se situe dans la recherche d'une bonne répartition de l'ensemble des valeurs possibles des arguments en entrée et dans le choix des représentants.
La réalisation des tests ne s'effectue généralement pas à la fin du développement d'une application mais s'échelonne tout au long des étapes du cycle de vie de cette application. Les tests pourront, par exemple, être construits lors de la spécification, de la conception et de l'intégration des différents programmes.

4.1. Tests construits à partir des spécifications

A partir des spécifications, on essaie de dégager un certain nombre de classes d'équivalence (identifiées en s'intéressant soit aux propriétés des entrées, soit aux relations de cause d'effet énoncées dans les spécifications).
On détermine alors un jeu de test qui couvre chacune des classes, ainsi que les cas limites de ces classes. Ensuite, on définit les résultats attendus pour les jeux de test choisis. Ces tests sont généralement appelés "tests black box".

4.2. Tests construits à partir de l'algorithme

Un algorithme peut être parcouru de son entrée à sa sortie par différents chemins (une condition de sélection simple définit 2 chemins distincts). Pour construire ces jeux de test appelés "white box", on va essayer de découvrir un ensemble de chemins représentatifs de l'ensemble des chemins possibles associés à l'algorithme. Pour ce, on peut, par exemple, choisir l'ensemble des chemins couvrant toutes les combinaisons possibles des issues de condition de sélection de l'algorithme ainsi que 0 ou 1 itération de chaque boucle. Ensuite, on choisira des données de tests pour chaque chemin.

4.3. Tests d'intégration

Les deux types de tests proposés ci-dessus ne visent qu'à détecter des erreurs au sein d'un programme considéré isolément. Souvent, ce programme fait partie d'un ensemble de programmes, on ne peut donc se contenter des 2 types de tests cités ci-dessus.
Il faudra en plus tester la compatibilité des interfaces, la coopération entre programmes...c'est l'objectif des tests d'intégration.

La liste des tests que l'on peut réaliser ne se réduit pas à ces 3 types de tests. D'autres tests tels des tests de performances, de sécurité, d'acceptation, d'ergonomie, d'installation...peuvent également être menés.

D E U X I E M E P A R T I E

APPLICATION

I N T R O D U C T I O N

Dans cette deuxième partie, nous développerons une application selon les principes exposés au chapitre 1 de la première partie de ce travail. Cette application a déjà été résolue à la C.G.E.R.

Dans le chapitre consacré à la construction des programmes (chapitre 2), nous ne nous limiterons pas à appliquer les principes du chapitre 1 de la première partie. Nous construirons les programmes, aussi, en utilisant la méthode de Jackson essentiellement parce que celle-ci semble être de plus en plus utilisée à la C.G.E.R. et que nous voulons évaluer (dans la 3ème partie) dans quelle mesure ces deux méthodes peuvent être complémentaires.

CHAPITRE 1 - S P E C I F I C A T I O N

1. DESCRIPTION DE LA PROCEDURE DOM80

1.1. Présentation générale

La procédure DOM80 est une procédure de domiciliation plus souple que les procédures habituelles et mieux adaptée à certaines situations réelles du genre suivant. Considérons le cas d'un organisme public (RTT,...) (appelé créancier) fournissant en permanence, à un nombre élevé de personnes (appelées débiteurs), un service de coût variable. Afin d'être payé de ses services, le créancier pourrait envoyer régulièrement à ses débiteurs des factures que ceux-ci règleraient en faisant verser le montant des factures sur le compte du créancier (par virement, chèque ou tout autre moyen du même genre).
Cette façon de procéder présente des inconvénients aussi bien pour le créancier que pour le débiteur ; par exemple cela occasionne des frais d'envoi au créancier ; cela oblige le débiteur à émettre un ordre de virement ou à se déplacer à sa banque pour retirer les fonds nécessaires au paiement des factures. Pour pallier ces inconvénients, le système financier belge offre une autre façon de procéder baptisée DOM80. Cette procédure permet au créancier d'obtenir un encaissement automatique de ses factures, quels qu'en soient le montant ou la périodicité, par l'intermédiaire de son organisme bancaire (appelé institution centralisatrice) et de ceux de ses débiteurs (appelés institutions domiciles). Cette procédure offre, sans nul doute, de nombreux avantages pour toutes les parties concernées mais il ne nous est pas possible de les détailler ici, essentiellement, parce que nous ignorons de nombreuses caractéristiques du système bancaire et du fonctionnement interne des banques. Toutefois les personnes qui ont imaginé cette procédure ont du s'assurer explicitement de son intérêt réel. En fait ce sont ces personnes qui ont pris les décisions essentielles pour la "programmation" des systèmes informatiques associés à la procédure DOM80.

1.2. Description sommaire de la procédure DOM80

La procédure DOM80 suppose un échange d'informations sur supports normalisés (bandes magnétiques, disquettes, réseau

téléphonique,...) entre les banques. Cet échange est réalisé via un organisme centralisateur appelé le C.E.C. (centre d'échange d'opérations à compenser). Pour adhérer à la procédure DOM80, un créancier en fait la demande auprès d'un de ses organismes bancaires (institution centralisatrice de ce créancier). Celui-ci accepte ou non la demande. S'il l'accepte, il communique au C.E.C. la description du nouveau créancier. Le C.E.C. met à jour la liste des créanciers affiliés à DOM80 (et affecte, notamment un numéro d'identification de créancier au nouveau promu). Chaque semaine il communique à tous les membres du C.E.C. cette liste up-to-date. Afin que le créancier puisse utiliser la procédure pour le règlement de ses factures, il faut qu'un accord soit conclu avec les débiteurs et leur organisme financier (institution domicile). Cet accord est notifié par un avis de domiciliation, dont chacune des parties détient un exemplaire garantissant le respect de l'accord par les deux autres. Dès lors, le créancier peut utiliser la procédure DOM80 pour l'encaissement automatique de ses factures et cela jusqu'à annulation de l'accord (révocation) sur décision d'une des trois parties. L'encaissement d'une facture peut être découpé en une ou plusieurs opérations en faveur du créancier (recouvrement) ou en faveur du débiteur (remboursement).
Le créancier enverra, à sa convenance, à son institution centralisatrice des supports normalisés contenant la description d'un ensemble d'opérations de ce genre.
L'institution centralisatrice se chargera de régler les opérations via le C.E.C. avec les institutions domiciles.

1.3. Faits et définitions

Tout créancier, affilié à DOM80, est identifié par un numéro d'identification de créancier. Ce numéro respecte le format suivant : *********-** (rmq : ce numéro est affecté par le C.E.C. selon une procédure bien précise, que nous ne décrirons pas ici car c'est inutile pour le problème qui nous occupe).

Un avis de domiciliation se présente comme indiqué à l'annexe 1. Il notifie l'accord suivant appelé contrat de domiciliation passe entre le débiteur (décrit à la case 1), le créancier (décrit à la case 2 et 8), l'institution domicile (décrit à la case 4) et le payeur (décrit à la case 6 ou à la case 1 si la case

6 n'est pas remplie) : les factures envoyées par le créancier au débiteur, portant le numéro de référence mentionné à la case 3 seront réglées selon la procédure DOM80, par l'institution domicile, au moyen du compte du payeur (case 5). Les avis de domiciliation sont identifiés par leur numéro de domiciliation (case 7). L'accord prend cours à la date mentionnée à la case 9. (*)

Il peut perdre son effet à une date ultérieure, appelée date de révocation.

La description au sens strict d'une opération relative à un avis de domiciliation est constituée des informations suivantes :

- le numéro d'identification du créancier,
- le numéro de domiciliation,
- le numéro de référence

  tels qu'ils figurent sur l'avis de domiciliation ;
- le numéro de compte du créancier,
- le nom du créancier,
- la date pivot,
- le montant de l'opération,
- la nature de l'opération (recouvrement ou remboursement),
- la communication au payeur première partie,
- la communication au payeur seconde partie,

  ces communications permettent au payeur d'identifier l'opération qui lui sera notifiée par un avis de débit ou crédit ;
- la provenance de la description, c'est-à-dire :

  soit le numéro d'identification du C.E.C. qui est alors aussi l'origine de la description,
  soit le numéro d'identification de la société chargée par un ou plusieurs créanciers de remettre les descriptions d'opérations à l'institution financière, cette société est appelée le remettant et dans ce cas le créancier est l'origine de la description.

La description au sens strict exclut toute information technique et organisationnelle. Pour désigner une description munie de telles informations on parlera de description au sens large.
Par la suite, nous utiliserons assez souvent la notion de description complétée. Cette notion n'a de sens que pour les descriptions d'opérations destinées à la C.G.E.R., c'est-à-dire

(*) Pour autant que le créancier ne renvoie pas l'avis qu'il a reçu, indiquant par là qu'il refuse l'accord ; en fait, l'accord prend cours à partir du moment où l'institution domicile a enregistré les informations relatives à ce contrat.

celles pour lesquelles les trois premiers chiffres du numéro de domiciliation = 048. Cette description complétée d'opérations a un sens différent suivant qu'il s'agit d'une opération acceptable ou non. Cette notion sera définie au paragraphe 3.5.
Une opération relative à un avis de domiciliation est soit un remboursement soit un recouvrement. Un recouvrement (remboursement) consiste à transférer un montant du compte du payeur (créancier) au compte du créancier (payeur).
La description d'une opération est remise par le créancier à son institution centralisatrice. Celui-ci précise, dans la description, la date à laquelle il désire que cette description soit mise à la disposition des institutions domiciles ; cette date est appelée la date pivot. (*)
Cette date pivot servira à déterminer la date de compensation, c'est-à-dire, la date à laquelle à lieu la comptabilisation des dettes et créances respectives dans les chambres de compensation.
Cette date de compensation sera calculée de la manière suivante : en cas de recouvrement, la date de compensation est la date pivot plus 4 jours ouvrables bancaires ; en cas de remboursement, la date de compensation est la date pivot elle-même.
Le compte du débiteur sera en cas de recouvrement débité entre la date pivot et la date de compensation ; en cas de remboursement il sera crédité à la date pivot qui est aussi la date de compensation.
Le compte du créancier sera en cas de recouvrement crédité peu après la date de compensation ; en cas de remboursement il sera débité peu après la date pivot.
Rmq : tout transfert ne s'effectue pas nécessairement de cette sorte ; en cas de recouvrement, l'institution centralisatrice accepte parfois de créditer le compte créancier avec une date valeur = date de compensation.

2. PROBLEME DU CONTROLE

Selon les modalités de fonctionnement de DOM80, l'institution domicile reçoit un ensemble d'opérations qu'elle a ordre de régler.

(*) Pour empêcher que les descriptions ne soient remises trop tôt et n'entrainent un transfert prématuré, la date pivot devrait être supérieure ou égale à la date d'exigibilité de la facture.

Cet ordre résulte d'un contrat de domiciliation signe avec le débiteur-payeur et le créancier. Avant de régler une opération, l'institution domicile vérifie la validité des opérations non seulement pour éviter que le créancier et le débiteur-payeur n'en contestent le règlement mais aussi tout simplement dans le souci d'effectuer des traitements sur des données valides (*).

Cete notion d'opération valide au sens intuitif du terme est une notion impossible à définir précisément en toute généralité. On ne pourra jamais énumérer toutes les propriétés puisqu'on ne les connait pas toutes à priori et même si on le prétend et qu'on les énumère effectivement rien ne permettra d'affirmer que cet ensemble de propriétés constitue la définition d'une opération valide. Tout ce que l'on peut faire c'est donner une définition qui nous semble se rapprocher le plus possible de la notion intuitive. A cet effet, la procédure DOM80 (basée sur l'existence d'un contrat de domiciliation) suggère qu'une description d'opération soit correcte ou valide :
si elle est compatible avec un contrat de domiciliation en cours (point 1) ;
si les différentes parties ont respecté les clauses du contrat (point 2) ;
si la situation du compte du payeur permet le règlement de cette opération (point 3).

Le but de l'institution domicile étant de vérifier la validité (au sens intuitif) des opérations, nous admettrons maintenant que cette validité se réduit à la définition donnée ci-dessus. Celle-ci est cependant encore trop générale par certains aspects. Aussi dans le but de la préciser, nous allons reprendre chaque point de cette définition en la détaillant et en se limitant à des conditions de validité "raisonnablement vérifiables". Les choix et les limitations ainsi décidés devront être justifiées.
Dans la mesure du possible on s'efforcera de proposer des conditions de validité automatisables mais on ne pourra exiger qu'elles le soient toutes. En effet, la notion intuitive de validité relevant du cas d'espèce est non effective ; et réduire l'ensemble des conditions de vérification à un ensemble de conditions automatisables

(*) Le fait que ce type de contrôle soit à charge de l'institution domicile est un choix quelque peu arbitraire dont la justification la plus probable est qu'une institution financière a plus confiance en un contrôle qu'elle réalise elle-même.

nous aurait trop éloignés de la notion intuitive de validité. Les conditions automatisables seront suffisamment restrictives pour que l'ensemble des opérations acceptables automatiquement (c'est-à-dire par traitement automatique) soit inclus dans l'ensemble des opérations acceptables intuitivement. (Une opération acceptable par traitement automatique sera ainsi sûrement une opération acceptable intuitivement ; l'inverse n'étant pas forcément vrai). Ce traitement automatique entrainera parfois un rejet excessif d'opérations aussi l'ensemble des opérations refusées automatiquement sera réexaminé manuellement et une opération refusée (automatiquement) pourra quand même être considérée comme acceptable.
Pour ne pas perdre l'avantage du traitement automatique, si le nombre d'opérations rejetées automatiquement devenait trop élevé , on aurait intérêt à distinguer deux types de rejet rejet par précaution et rejet définitif. Seules alors les opérations rejetées par précaution devraient être réexaminées manuellement. Les critères permettant de faire une telle distinction parmi les rejets devraient être discutés entre les programmeurs du traitement automatique et les personnes responsables du traitement manuel.

Ceci étant dit, nous allons essayer de préciser les 3 points de la définition d'une description d'opération valide. Pour des raisons de clarté, nous préciserons le permier point de la définition en deux temps ; au premier on cernera la notion de compatibilité au second celle de contrat en cours.

a. Le contrôle de la compatibilité d'une description d'opération avec un contrat de domiciliation aurait certainement été plus complet mais aussi plus coûteux si l'on avait repris dans la description d'une opération l'ensemble des informations relatives au contrat de domiciliation. Face à ce dilemne, coût efficacité, on s'est limité à reprendre, dans une description, 3 informations relatives au contrat de domiciliation (le numéro d'identification du créancier, le numéro de domiciliation, le numéro de référence). Logiquement on peut donc déduire qu'une description sera compatible avec un contrat de domiciliation ssi ces 3 informations reprises dans un cdd et une description d'opération sont égales.

b. Dans tout ce qui vient d'être dit on a volontairement omis, par simplicité, de souligner le fait que le cdd doit être en cours ce que nous nous allons à présent tenter de définir. Tout d'abord

un cdd ne peut se trouver que dans deux états : en cours ou révoqué. On admettra qu'il est dans un de ces 2 états à partir de la date d'entrée en vigueur (dev) du contrat, c'est-à-dire la date à laquelle on enregistre les informations relatives à ce cdd.
De même la date de révocation (dr) sera la date où l'on a enregistré les informations relatives à une révocation du cdd.
A partir de ces 2 notions (dve et dr) on définit la période P durant laquelle le contrat est considéré comme en cours.

P = de dev jusqu'à dr

Intuitivement, la compatibilité d'une description d'opération avec un contrat de domiciliation devrait être recherchée parmi les contrats en cours pendant une période couvrant celle où le service (justifiant l'exécution de cette opération) a été presté. Ceci est infaisable puisque nous ne disposons pas d'informations concernant la période pendant laquelle le service est fourni.
Pratiquement on se limitera à rechercher la compatibilité d'une description d'opération avec un contrat de domiciliation en cours au moment où l'on effectue le contrôle. Cette limitation entrainera le rejet d'un certain nombre d'opérations qui intuitivement auraient dû être acceptées. Pour atténuer ce rejet ces opérations pourront être acceptées lors de la vérification manuelle.
A présent, nous allons préciser le deuxième point de la définition : "les différentes parties ont respecté les clauses du contrat".
Apparemment il n'existe aucun moyen infaillible pour s'assurer du respect total des clauses du contrat. Aussi, on offrira à chacune des parties la possibilité d'émettre une opposition sur l'exécution du contrat ; on supposera que l'absence d'opposition signifie que le contrat a été respecté et que les différentes parties en sont satisfaites.
L'existence d'une opposition sur un contrat impliquera un rejet soit définitif soit par précaution (des opérations liées à ce contrat).
Le type de rejet impliqué par l'existence d'une ou plusieurs oppositions sera donc discuté entre les programmeurs et les personnes responsables du traitement manuel et les personnes responsables de l'acceptation des oppositions.
Le troisième point de la définition ne sera pas développé ici. Il correspond à un contrôle réalisé (pour toute opération domiciliée ou non) au service comptabilisation. Pour éviter toute redondance, et dans le souci de simplicité, ce contrôle restera à charge du service comptabilisation. Le rôle du service domiciliation se limitera à contrôler de manière automatique le respect des points 1

et 2 de la définition d'une opération valide. Par la suite, on utilisera le terme acceptable pour désigner une opération ayant satisfait à ces 2 points. Elle sera acceptable automatiquement si à l'issue des contrôles effectués au service domiciliation elle est acceptable. Elle sera acceptable manuellement si elle a été rejetée au service domiciliation, réexaminée manuellement et si les raisons de son rejet ont été finalement considérées comme non valables.

Ayant posé le problème du contrôle, nous allons à présent déduire les informations nécessaires à sa solution.
En ce qui concerne la vérification du premier point (comme il s'agit de vérifier la compatibilité entre une description d'opération et un contrat de domiciliation en cours) il faudra disposer d'un certain nombre d'informations sur ce contrat à savoir celles qui sont (ou devraient être) en commun avec la description d'opération, c'est-à-dire : le numéro d'identification du créancier, le numéro de domiciliation, le numéro de référence plus une information traduisant son état (en cours ou révoqué).
Concernant la vérification du deuxième point, on devra s'assurer qu'il n'existe aucune opposition sur le contrat, donc disposer d'une information traduisant l'existence d'une opposition sur un contrat.
La question de savoir comment ces informations seront organisées l'une par rapport à l'autre sera discutée ultérieurement.

3. IMPLEMENTATION

3.1. Partie de l'existant à modifier

Jusqu'à présent, on a posé le problème du contrôle en faisant le plus possible abstraction des contraintes informatiques et organisationnelles (*).
Si l'on veut aller plus loin et se demander comment on pourra implémenter une telle solution, il faudra les prendre en compte.
On fera l'hypothèse que cette solution s'insère dans un environnement où les procédures concernant DOM80 (au niveau interbancaires) ont déjà été instaurées. Ceci permet de diviser le problème

(*) Le plus possible mais pas totalement puisque pour savoir quelles sont les conditions raisonnablement vérifiables on a dû se baser sur une idée intuitive de ce qui est faisable à la C.G.E.R.

général de DOM80 et d'avoir une application relativement réduite à traiter.
Sous cette hypothèse, les contraintes de l'existant sont les contraintes liées au problème de DOM80. On ne les examinera pas toutes. On s'intéressera uniquement à la partie de l'existant susceptible d'être modifiée par l'implémentation de la solution au problème de contrôle. Cette solution répartit l'ensemble des descriptions d'opérations en deux sous-ensembles :

- un sous ensemble reprenant les descriptions d'opérations acceptables automatiquement et transmises avec d'autres informations à la comptabilisation ;
- un sous-ensemble reprenant les descriptions d'opérations non acceptables automatiquement pour lesquelles il faudra créer un traitement spécial.

Logiquement la partie à modifier sera celle qui, auparavant, se préoccupait de fournir à la comptabilisation les informations nécessaires à la comptabilisation des opérations. Quand on regarde l'existant, on s'aperçoit qu'il existe une partie du système chargée de traiter les descriptions d'opérations et de fournir aux différents services de la C.G.E.R. les renseignements nécessaires à leur activité. Nous ne nous attarderons pas à décrire toutes les fonctions de cette partie du système, car pour comprendre la raison de certaines fonctions, il aurait fallu avoir une connaissance plus approfondie du système (connaissance que nous n'avons pas). Aussi, on s'attachera uniquement à la fonction chargée de transmettre à la comptabilisation, les informations dont elle a besoin, (informations relatives aux opérations à comptabiliser) et on étudiera les répercussions apportées à cette fonction suite à l'instauration de la solution au problème du contrôle (*).

Avant de décrire de manière plus précise cette fonction, remarquons qu'une des contraintes organisationnelles de la C.G.E.R. impose de répartir les descriptions d'opérations suivant la gestion à laquelle elles sont destinées pour être comptabilisées. La gestion relative

(*) Cette restriction ne signifie pas qu'il n'y aura pas de répercussion sur les autres fonctions, seulement notre connaissance du système étant relativement réduite nous nous sentons incapables d'en discuter de manière satisfaisante.

à une description se déduit à partir du numéro de compte débiteur. Cette remarque étant faite, décrivons de manière plus précise la fonction qui nous intéresse. Elle reçoit en entrée des descriptions d'opérations destinées au C.E.C. ou à la C.G.E.R. et elle produit en sortie, pour les opérations destinées à la C.G.E.R uniquement de nouvelles descriptions d'opérations qu'elle répartit suivant la gestion à laquelle elles sont destinées (elle ne fait rien en ce qui concerne les descriptions d'opérations destinées au C.E.C.). Ces nouvelles descriptions se déduisent des anciennes en rajoutant pour chaque description d'opération le numéro de compte du débiteur et l'identification de la gestion.

3.2. Impact de la solution au problème du contrôle sur cette partie

Maintenant examinons quel sera l'impact sur cette fonction de la solution au problème du contrôle. On a vu que cette solution exigeait que les descriptions d'opérations à transmettre à la comptabilisation soient des descriptions d'opérations acceptables automatiquement.

L'effet d'une telle solution sur la fonction qui nous intéresse sera le suivant : en plus de sélectionner parmi les descriptions d'opérations fournies en entrée celles destinées à la C.G.E.R., elle devra s'assurer qu'il s'agit bien d'opérations acceptables automatiquement ; pour les descriptions d'opérations non acceptables automatiquement, il faudra créer un traitement spécial. Ces descriptions devant être réexaminées manuellement le traitement consistera à fournir une description qui soit satisfaisante aux yeux des personnes responsables de l'examen manuel.

Idéalement, la définition de ce qu'est une description satisfaisante devrait ressortir des entretiens avec les personnes responsables. Ceci ne nous est malheureusement pas possible, aussi on essaiera ci-dessous d'imaginer ce que pourrait bien être une description satisfaisante.

A l'issue du réexamen manuel, les descriptions d'opérations pourront être acheminées à la comptabilisation, il faut donc qu'à l'entrée du réexamen manuel l'on dispose de toutes les informations relatives aux opérations à comptabiliser, c'est-à-dire leur description augmentée du numéro de compte du débiteur et de l'identification de la gestion (*).

(*) Pour autant que cela soit possible : si le numéro de domiciliation n'existe pas, on ne pourra déduire ces deux informations, on dira alors qu'elles ne sont pas déductibles.

D'autre part, le but de ce réexamen manuel étant de décider si le refus d'une opération est justifié ou non, il est nécessaire de mentionner dans la description satisfaisante d'une opération le ou les raisons de celui-ci.
Quels sont les différent motifs de refus ; quels sont ceux que l'on inclut dans la description satisfaisante et pourquoi ?
C'est à ces questions que l'on va maintenant tenter de répondre.
On a vu plus haut, qu'il y a deux raisons majeures pour qu'une description d'opération soit non acceptable automatiquement.
La première raison est qu'il n'existe aucun contrat de domiciliation en cours compatible avec la description d'opérations ; la seconde est qu'au moins une des parties n'ait pas respecté les clauses du contrat. Se contenter de mentionner ces 2 raisons dans la description d'une opération ne nous semble pas satisfaisant. Ces raisons sont en effet trop générales, insuffisantes pour permettre de discuter du bien fondé du rejet ; elles nécessitent d'être détaillées.
Nous détaillerons, tout d'abord, les raisons qui amènent à un rejet d'une opération pour cause de non compatibilité avec un contrat de domiciliation. Dans la mesure du possible, on essaiera de les présenter selon un ordre logique (ordre décroissant d'importance).
Auparavant remarquons que nous supposons que l'existence des renseignements relatifs à un créancier (son numéro d'identification notamment) est indépendant de l'existence des informations relatives à un contrat de domiciliation (l'inverse n'est pas vrai).
Cette hypothèse nous semble justifiée pour les raisons suivantes : un créancier peut exister, demander son affiliation à DOM80 et recevoir un numéro d'identification sans que pour autant il ait déjà conclu des contrats de domiciliation ; par contre l'existence d'un contrat de domicliation suppose l'existence et l'affiliation à DOM80 du créancier concerné par ce contrat.
C'est cette hypothèse qui justifiera le fait que la question de compatibilité entre un cdd et une description d'opération ne peut logiquement être posée qu'après s'être assuré que d'une part le numéro de domiciliation existe et que d'autre part le numéro d'identification du créancier existe.
Le numéro de domiciliation existant, on se demandera :
- si le contrat est en cours,
- si il n'y a pas d'opposition sur le cdd,
- si le numéro d'identification du créancier mentionné dans la description d'opération est égal à celui mentionné dans le cdd,

- si le numéro de référence mentionné dans la description d'opération est égal à celui mentionné dans le cdd.

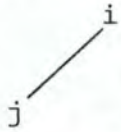
Concernant la seconde raison majeure entrainant le rejet d'une opération, le seul motif apparaissant est l'existence d'une opposition. On distinguera 2 types d'opposition : opposition sur un contrat, opposition sur un créancier. Ce dernier type d'opposition a été introduit pour permettre de bloquer la partie créancier par exemple suite à une faillite de ce créancier, ou parce que le créancier se rend compte qu'il a remis des descriptions d'opérations comportant des données erronnées ou encore parce qu'il exige continuellement des montants trop élevés...

En résumé, il existe 7 motifs de refus provoqués par le non respect des 7 conditions suivantes :

1. le numéro d'identification du créancier existe,
2. il n'y a pas d'opposition sur ce créancier,
3. le numéro de domiciliation existe,
4. le contrat est en cours,
5. il n'y a pas d'opposition sur le contrat,
6. les numéros d'identification du créancier mentionnés dans la description d' opération et dans le cdd sont compatibles,
7. les numéros de référence mentionnés dans la description d'opération et dans le cdd sont compatibles.

Le non respect de ces 7 conditions ne peut se produire simultanément ; en fait, au maximum 5 motifs de refus peuvent apparaître simultanément :

Signification des flèches :



le fait de savoir si la condition j est respectée n'a de sens que si la condition i l'est

Pour aller plus loin et se rapprocher de l'étape de programmation au sens strict, on pourrait maintenant discuter :

a. de l'organisation physique des informations nécessaires au contrôle ;

b. de la représentation des entrées du programme contrôle ;

c. de la représentation des sorties du programme contrôle.

3.3. Organisation physique des informations nécessaires au contrôle

Nous préférerons reporter cette discussion le plus loin possible. Ainsi nous supposerons que l'on dispose de modules à l'intérieur desquels ces questions sont résolues. Nous créerons les modules suivants :

EXICDD

Spécification :
Reçoit en entrée un numéro de domiciliation W-NU-DOMI et renvoie en sortie une variable booléenne CDDEXIST ayant la valeur 1 s'il existe un contrat de domiciliation identifié par ce numéro de domiciliation ; 0 sinon.
Dans le cas où la valeur est 1, il renverra en plus dans les variables IDENTCRE, NUREF, OPPCDD, ETATCDD, W-DATE-REVOCATION, W-DESC-OP-CDD, CPTE-DEB, respectivement le numéro d'identification du créancier, le numéro de référence repris dans le cdd, la présence ou non d'une opposition (la valeur 0 traduit l'absence d'opposition, la valeur 1 la présence), l'état du contrat (la valeur 0 pour l'état en cours, la valeur 1 pour l'état révoqué), la date de révocation si le contrat est révoqué, la description de l'opposition s'il y a une opposition sur ce contrat, et le numéro de compte débiteur.

EXICRE

Spécification :
Reçoit en entrée un numéro d'identification de créancier W-NU-ID-CRE et renvoie en sortie une variable booléenne CREEXIST ayant la valeur 1 s'il existe un créancier identifié par ce numéro d'identification de créancier, 0 sinon.

Dans le cas où la valeur est 1, il renverra en plus dans la variable OPPCREANCIER la valeur 1 s'il y a une opposition sur le créancier, la valeur 0 sinon et renverra dans la variable W-DESC-OP-CRE la description de l'opposition sur le créancier s'il y en a une.

3.4. Représentation des entrées

Concernant la représentation des entrées, il n'y a pas lieu d'en discuter. La seule représentation possible est imposée par la procédure DOM80.
L'entrée est un ENSEMBLE DE DESCRIPTIONS D'OPERATIONS au sens strict représenté par un fichier FICH-IN selon les conventions de représentations suivantes : cet ensemble est partitionné en sous-ensembles de descriptions ayant en commun 2 informations : la date pivot et l'origine de la description (C.E.C. ou créancier). Ces sous-ensembles sont représentés par des fichiers logiques (F.L.). Un FICHIER LOGIQUE est une séquence d'enregistrements consitutée d'un enregistrement début, d'une suite d'enregistrements opération et d'un enregistrement fin.
L'ENREGISTREMENT DEBUT (ED) reprend les informations communes aux descriptions, c'est-à-dire :
si la description est originaire du C.E.C. :
la date pivot,
l'origine (= provenance = n° d'identification du CEC) ;

si la description est originaire d'un créancier :
la date pivot,
l'origine (n° d'identification du créancier),
la provenance (n° d'identification du remettant),
le numéro de compte créancier,
+ des informations techniques.

L'ENREGISTREMENT OPERATION (EOP) reprend pour chaque description :
le numéro de domiciliation,
la nature de l'opération,
le montant de l'opération,
le nom du créancier,
la communication première partie,
la communication seconde partie,
le numéro de référence

+ des informations techniques.

+ Si la description est originiaire du C.E.C. :

le numéro de compte créancier,

le numéro d'identification du créancier.

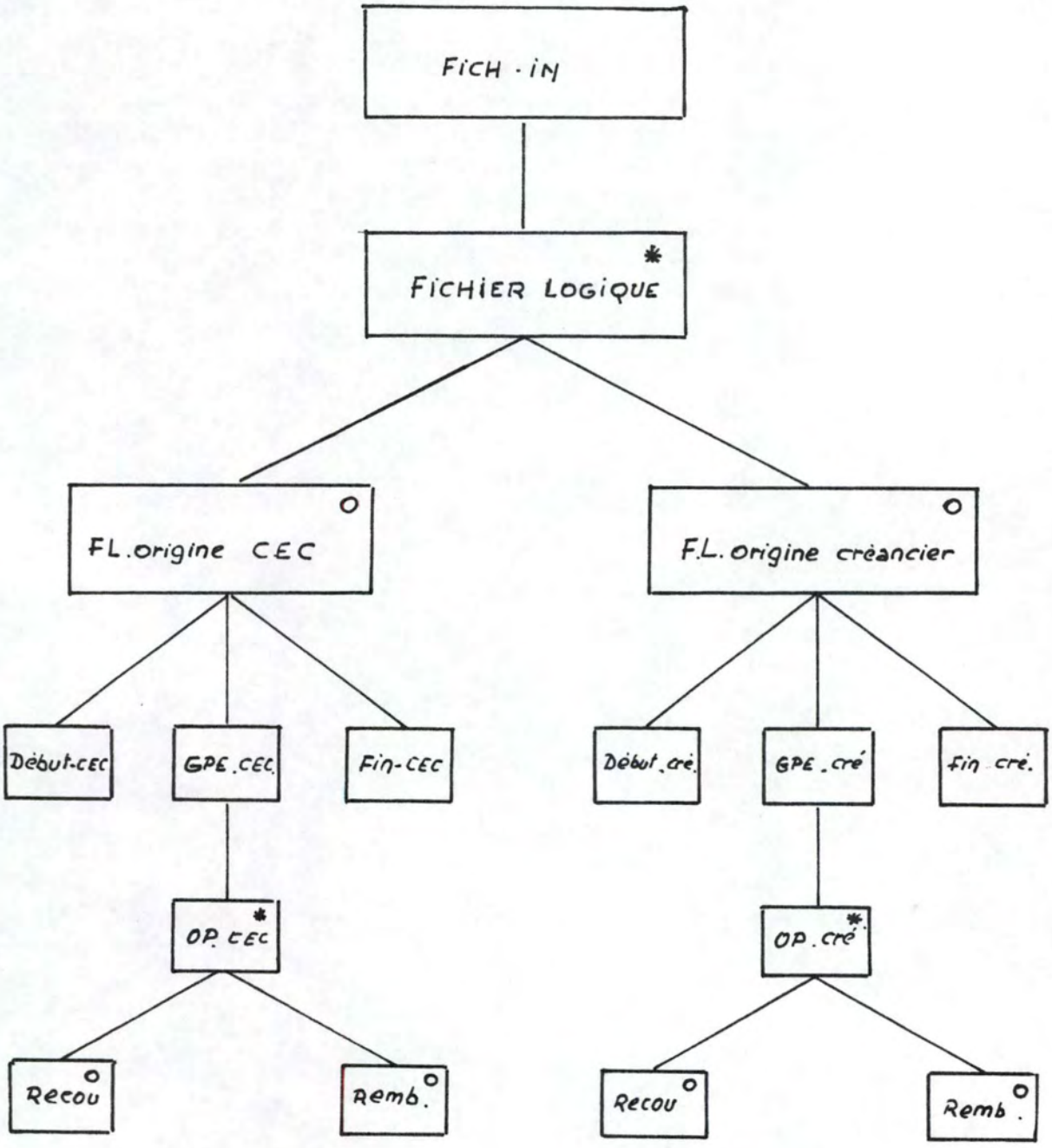
L'ENREGISTREMENT FIN reprend des informations récapitulatives.

Tableau récapitulatif de la répartition des informations (constituant la description au sens strict) entre enregistrement début (ED), enregistrement opération (EOP)

Informations contenues dans une description au sens strict	desc. orig. C.E.C.		desc. orig. créancier	
	ED	EOP	ED	EOP
1. Date pivot	*		*	
2. numéro d'identification créancier		*	*	
3. n° compte créancier		*	*	
4. nom de créancier		*		*
5. nature de l'opération		*		*
6. montant de l'opération		*		*
7. numéro de domiciliation		*		*
8. numéro de référence		*		*
9. communication première partie		*		*
10 communication seconde partie		*		*
11. la provenance	*		*	

Les enregistrements opérations du fichier logique cec sont triés sur le numéro d'identification du créancier, puis sur le numéro de domiciliation, tandis que les enregistrements opérations appartenant au fichier logique originaire d'un créancier sont triés sur le numéro de domiciliation.

Description de la représentation des entrées



3.5. Représentation des sorties

Nous supposerons que le programme contrôle génère des informations destinées à différents services sans se préoccuper du format exigé par ces services.

Les sorties seront représentées par deux fichiers :

- le fichier des opérations acceptables (FO-ACC) formé d'enregistrements (ENREG-ACC) représentant chacun la description complétée d'une opération acceptable ;
- le fichier des opérations non acceptables (FO-NONACC) formé d'enregistrements (ENREG-NONACC) représentant chacun la description complétée d'une opération non acceptable.

Une description complétée d'opération acceptable est constituée :

- de la description au sens strict d'une opération,
- du numéro de compte débiteur,
- du numéro d'identification de la gestion.

Une description complétée d'opération non acceptable est constituée :

- de la description au sens strict d'une opération,
- du numéro de compte débiteur,
- du numéro d'identification de la gestion (les zones devant contenir ces deux dernières informations seront tout simplement mises à blanc si la description au sens strict ne permet pas de les déduire),
- d'un tableau reprenant dans l'ordre mentionné page 92 le résultat des 7 contrôles, chaque élément de ce tableau peut prendre la valeur 0 ou 1

  $\forall$ i : 1 <= i <= 7

  tab (i) = 0 signifie que le résultat du ième contrôle est positif

  tab (i) = 1 signifie que le résultat du ième contrôle est négatif

  tab (i) est indéfini si le ième contrôle n'a pas de sens ;
- le numéro d'identification du créancier mentionne dans le contrat de domiciliation, s'il est incompatible avec le no d'identification indiqué dans la description,
- le numéro de référence, mentionné dans le contrat de domiciliation, s'il est incompatible avec le no de référence indiqué dans la description (le contenu des zones devant renfermer ces deux dernières informations est quelconque si le résultat des contrôles de la compatibilité des numéros d'identification du créancier et de référence repris dans le contrat de domiciliation avec ceux repris dans la description est positif),
- la date de révocation si le cdd est révoqué,

- la description de l'opposition sur le créancier s'il y a une opposition sur ce créancier,
- la description de l'opposition sur le cdd s'il y a une opposition sur ce cdd

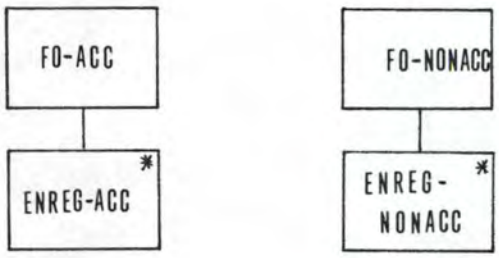


figure 3.5

3.6. Spécification du programme contrôle

- reçoit en entrée un ensemble (FICH-IN) de descriptions (destinées au C.E.C. ou à l'institution domicile C.G.E.R.) représenté selon les conventions citées ci-dessus au paragraphe 3.4 ;
- génère en sortie deux fichiers :
  * un fichier (FO-ACC) reprenant, pour les descriptions d'opérations acceptables destinées à l'institution domicile : C.G.E.R., leur description complétée,
  * un fichier (FO-NONACC) reprenant pour les descriptions d'opérations non acceptables destinées à l'institution domicile C.G.E.R., leur description complétée,

et ce, selon les conventions de représentation des sorties données au paragraphe 3.5.

Schématiquement, en représentant un fichier par un cercle et un programme par un rectangle, on a :

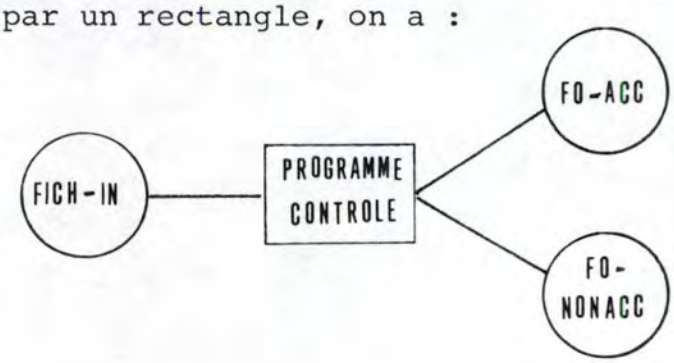


Figure 3.6

CHAPITRE 2 - CONSTRUCTION DU PROGRAMME CONTROLE

Dans ce chapitre, nous construirons le programme contrôle selon deux méthodes dont les principes ont été exposés dans la première partie de ce travail.

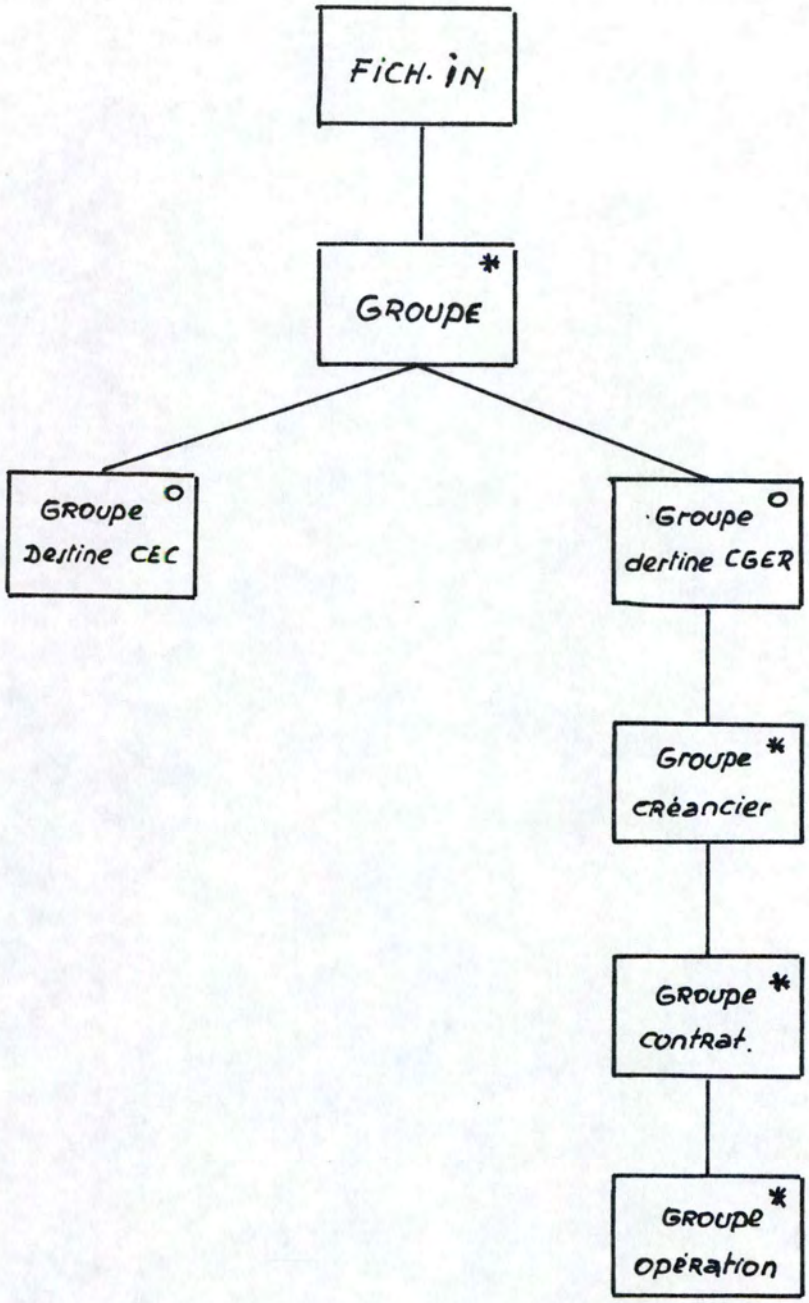
1. CONSTRUCTION SELON LA METHODE DE JACKSON

1.1. Spécification du programme contrôle

Voir 3.6. du chapitre 1.

1.2. Structuration des entrées et sorties du programme contrôle

Il s'agit, tout d'abord, de donner une description des données en entrée et en sortie. Il aurait fallu reprendre ci-dessous la description de FICH-IN, FO-ACC et FO-NONACC telle que décrite dans le programme COBOL ; pour ne pas allourdir inutilement le texte, nous ne l'avons pas fait.
Ensuite, il nous faut définir la structure des entrées et des sorties. Cette description doit refléter autant que possible notre compréhension du problème ; c'est la raison pour laquelle elle diffère de la description donnée aux paragraphes 3.4, 3.5 du chapitre 1. Le problème qui nous préoccupe est de vérifier si les opérations destinées à la C.G.E.R. sont acceptables ou non. Cette vérification nécessite d'effectuer 7 contrôles : 2 au niveau du créancier (contrôles 1 et 2 p 92), 4 au niveau du contrat (contrôles 3, 4, 5, 6), 1 au niveau de l'opération (contrôle 7).
Pour ne pas effectuer plusieurs fois un même contrôle sur une même donnée, l'idéal aurait été que les descriptions d'opérations destinées à la C.G.E.R. soient regroupées (en entrée) par créancier et puis par contrat.
On aurait pu structurer le fichier d'entrée de cette sorte :



Pour obtenir une telle structure, il aurait fallu réorganiser totalement le fichier d'entrée.

On a préféré partir du fichier des entrées tel qu'il nous était fourni et essayer de le représenter par une structure reflétant le plus possible la structure idéale. De cette façon, on a obtenu la structure suivante :

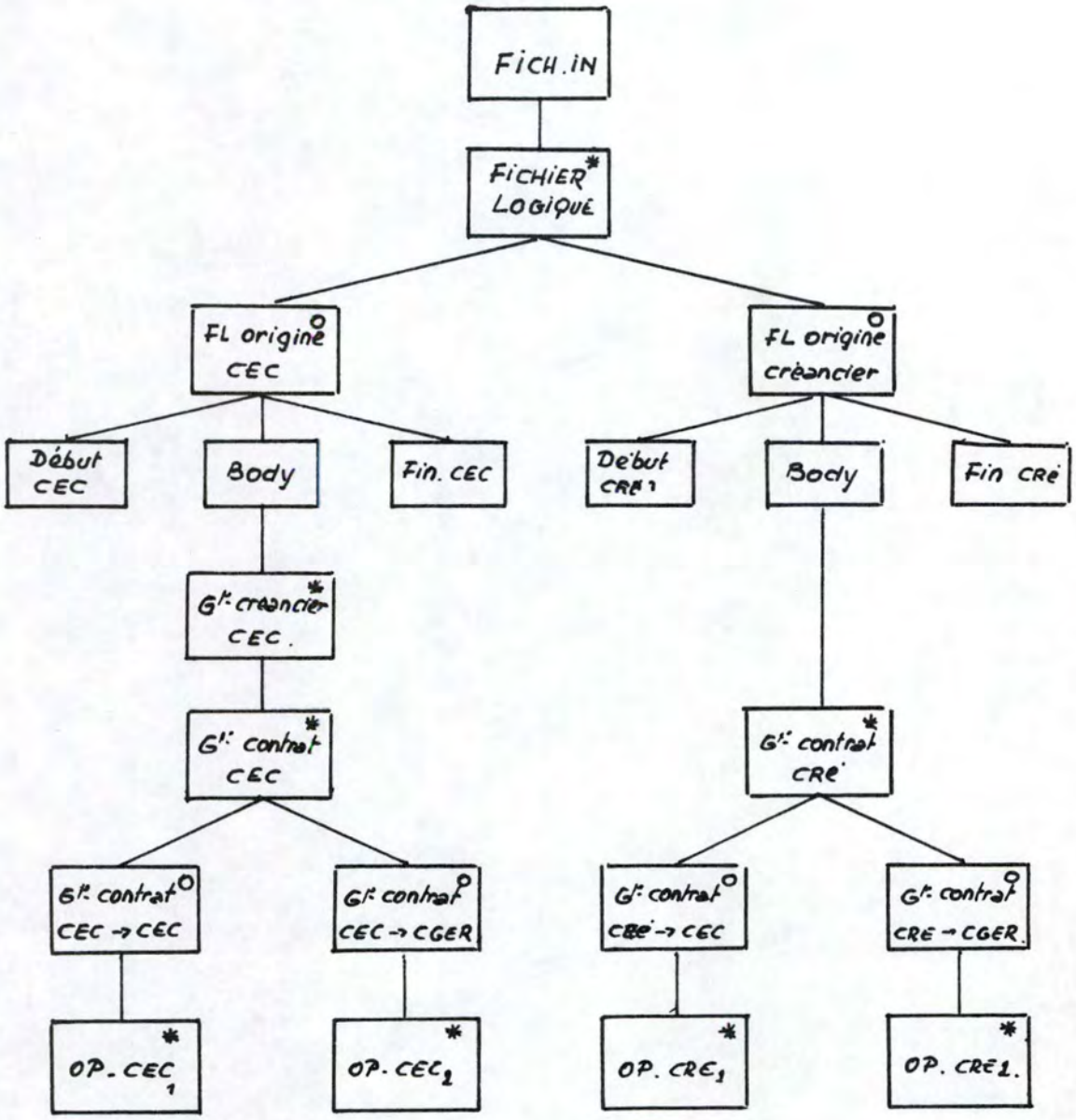
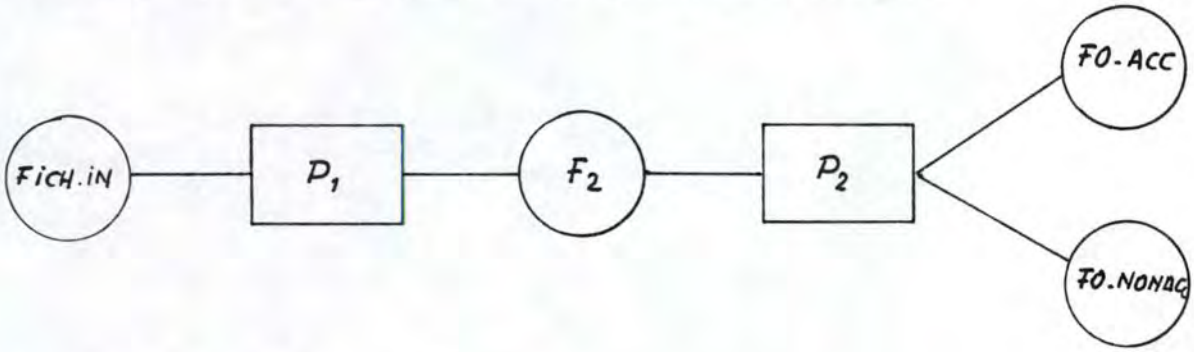


Figure 1.2.2.

On constate que dans cette structure, il existe déjà des notions de groupe créancier, groupe contrat, mais qu'ils se situent à des niveaux différents suivant qu'il s'agit d'un groupe de descriptions originaires de C.E.C. ou d'un créancier.

1.3. Recherche des correspondances - Conflit de structure et solution

Il semble évident qu'il soit impossible de trouver une correspondance entre les structures des entrées (figure 1.2.2 page 101) et des sorties (figure 3.5. page 98). On se trouve donc en présence d'un conflit de structure. Il sera résolu en créant un fichier intermédiaire F2 sur lequel on pourra placer 2 structures en correspondance l'une avec la structure des entrées, l'autre avec la structure des sorties (Ce fichier ne reprendra du fichier d'entrée que les descriptions destinées à la C.G.E.R.).
Sur base des 2 correspondances établies, nous construirons 2 programmes P1 et P2 reliés entre eux à l'aide de F2.
Ces 2 programmes remplacent le programme contrôle



1.4. Spécification de P1

- reçoit en entrée un ensemble de descriptions au sens strict représenté par le fichier FICH-IN selon les conventions de représentation indiquées au paragraphe 3.4 du chapitre 1 ;

- génère en sortie l'ensemble des descriptions destinées à la C.G.E.R. représenté par le fichier F2 selon les conventions suivantes :

F2 est l'ensemble des descriptions d'opérations destinées à la C.G.E.R (de FICH-IN). Cet ensemble est partitionné en groupes (appelés groupes créancier) ayant en commun la date pivot, le numéro d'identification du créancier et la provenance.
Chacun de ces groupes créancier est représenté par un enregistrement entête-créancier suivi d'une suite de groupes contrat. Chacun des groupes contrat est représenté par un enregistrement entête-contrat suivi d'une suite d'enregistrements opération.
Un enregistrement entête-créancier reprend les informations communes aux descriptions d'un groupe créancier et est caractérisé par un numéro d'identification ayant la valeur 0. Un enregistrement entête-contrat reprend les informations communes aux descriptions

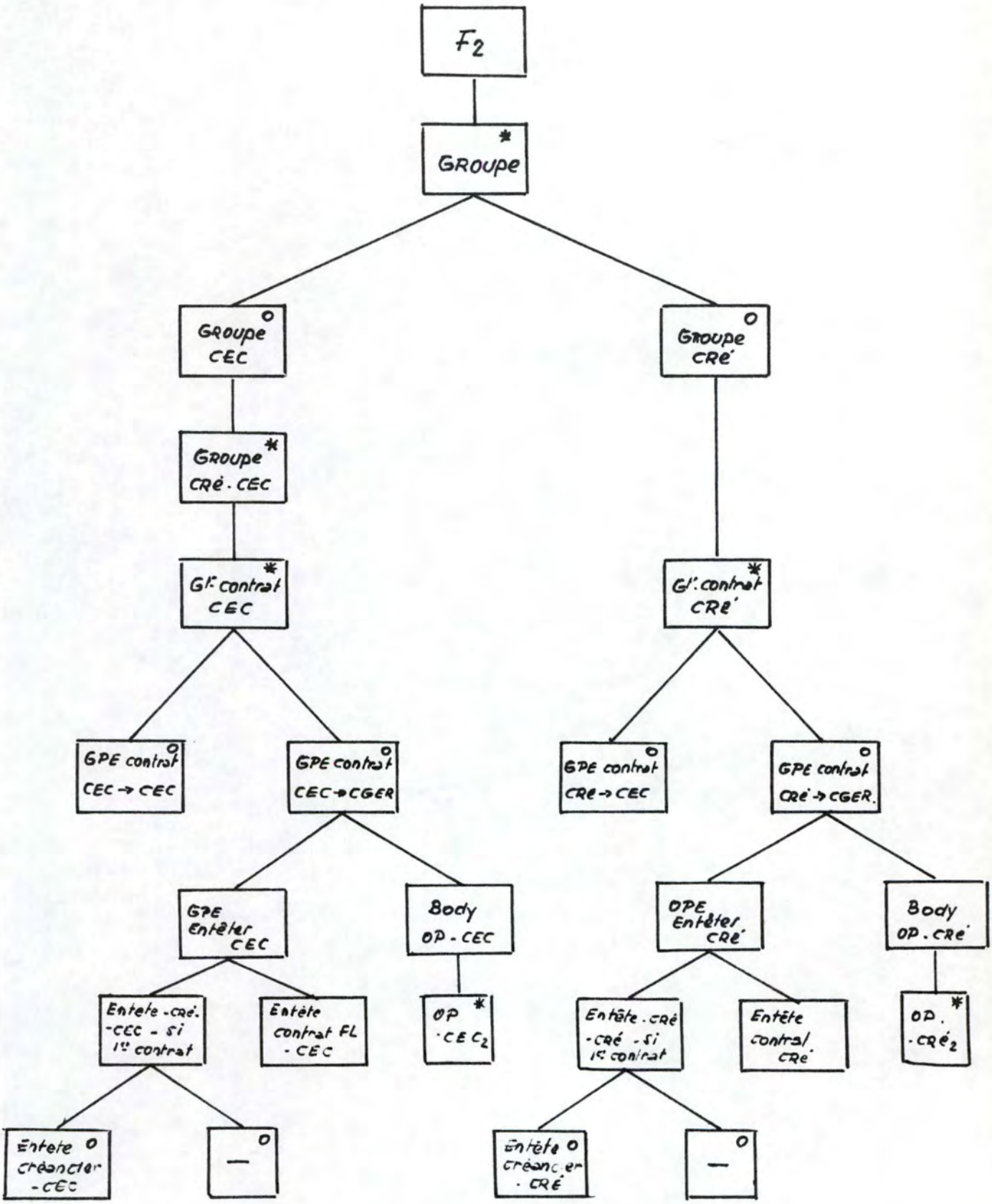
d'un groupe contrat à savoir le numéro de domiciliation, cet enregistrement est caractérisé par un numéro d'identification ayant la valeur 2. Un enregistrement opération reprend les informations suivantes : la nature et le montant de l'opération, le nom du créancier, la première et la seconde partie de la communication au payeur, le numéro de compte créancier. Cet enregistrement est caractérisé par un numéro d'identification ayant la valeur 1.

1.5. Structuration des entrées et sorties de P1

La structure des entrées est celle représentée par la figure 1.2.2. du paragraphe 1.2.

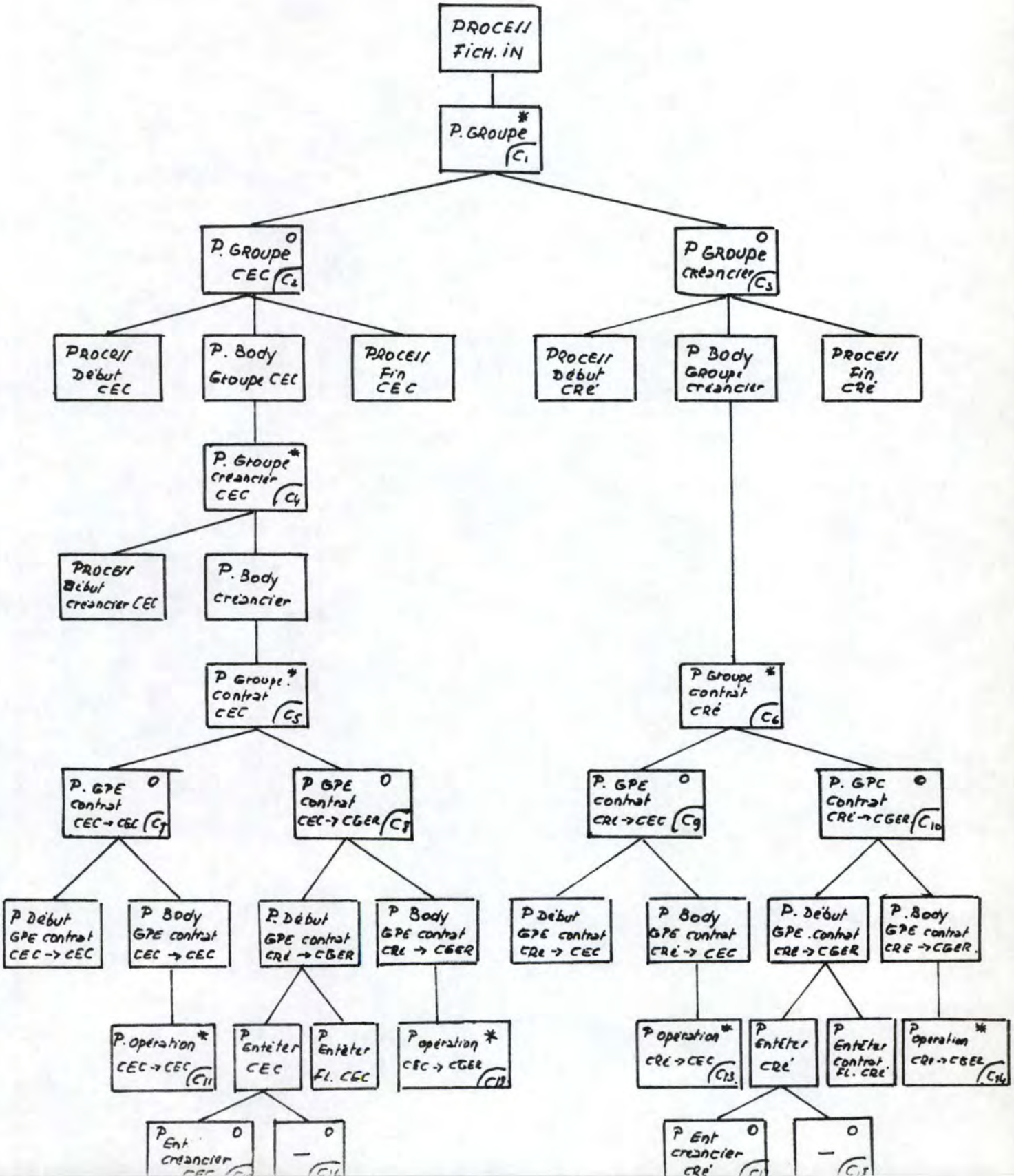
La structure des sorties est une structure pouvant être placée sur le fichier F2 et être mise en correspondance avec la structure des entrées. Selon cette structure, le fichier F2 est vu comme un ensemble de groupes originaires soit du C.E.C. (groupe C.E.C.) soit d'un créancier (groupe CRE).
Chaque groupe C.E.C. est vu comme une suite de groupes CRE-CEC.
Chaque groupe CRE-CEC (groupe CRE) est vu comme une suite de groupe contrat CEC (groupe contrat CRE).
Un groupe contrat CEC (groupe contrat CRE) est soit destiné à la C.G.E.R. soit destiné au CEC.
Un groupe contrat CEC (CRE) destiné à la C.G.E.R. est vu comme un enregistrement ENTETE-CREANCIER-CEC (CRE) si le groupe contrat-CEC (CRE) est le premier groupe contrat d'un groupe CRE-CEC (groupe CRE) (sinon rien) suivi d'un enregistrement entête contrat CEC (CRE) suivi d'une suite d'enregistrement $OP\text{-}CEC_2$ ($OP\text{-}CRE_2$).

Un groupe contrat cec (cré) destiné au cec est constitué d'une suite vide d'enregistrement $OP\text{-}CEC_2$ ($OP\text{-}CRE_2$).


F2
Groupe *
Groupe CEC o
Groupe CRé o
Groupe CRé . CEC *
Gl. contrat CEC *
Gl. contrat CRé *
GPE contrat CEC → CEC o
GPE contrat CEC → CGER o
GPE contrat CRé → CEC o
GPE contrat CRé → CGER. o
GPE Entêter CEC
Body OP - CEC
OPE Entêter CRé
Body OP . CRé
Entete - cré. - CEC - si 1er contrat
Entête contrat FL - CEC
OP · CE C2 *
Entête . cré - cré - si 1er contrat
Entête contrat/ CRé
OP . · CRé2 *
Entete o créancier - CEC
— o
Entête o créancier · CRé
— o

1.6. Recherche d'une correspondance entre les structures des entrées et des sorties de P1

En essayant de trouver une correspondance entre les structures des entrées et des sorties de P1, on est amené à élaborer l'arbre fusion suivant :



Certains des composants de cet arbre expriment l'existence d'une correspondance entre un composant de FICH-IN et un composant de F2. On les appellera composants correspondance.
Nous les avons repris dans le tableau suivant :

Composant correspondance	Composant de FICH-IN	Composant de F2
1. PROCESS FICH-IN	FICH-IN	F2
2. P GROUPE	FICHIER LOGIQUE	GROUPE
3. P-GROUPE-CEC	F.L. ORIGINE CEC	GROUPE-CEC
4. P-GROUPE-CREANCIER	F.L. ORIGINE CREANCIER	GROUPE CRE
5. P-GROUPE-CREANCIER-CEC	GROUPE-CREANCIER CEC	GROUPE-CRE-CEC
6. P-GROUPE-CONTRAT-CEC	GROUPE-CONTRAT-CEC	GROUPE-CONTRAT-CEC
7. P-GROUPE-CONTRAT-CRE	GROUPE-CONTRAT-CRE	GROUPE-CONTRAT-CRE
8. P-GPE-CONTRAT-CEC-DEST-CEC	GROUPE-CONTRAT-CEC-DEST-CEC	GROUPE-CONTRAT-CEC-DEST-CEC
9. P-GPE-CONTRAT-CEC-DEST-CGER	GROUPE-CONTRAT-CEC-DEST-CGER	GROUPE-CONTRAT-CEC-DEST-CGER
10. P-GPE-CONTRAT-CRE-DEST-CEC	GROUPE-CONTRAT-CRE-DEST-CEC	GROUPE-CONTRAT-CRE-DEST-CEC
11. P-GPE-CONTRAT-CRE-DEST-	GROUPE-CONTRAT-CRE-DEST-CGER	GROUPE-CONTRAT-CRE-DEST-CGER
12. P-OPERATION-CEC-DEST-CGER	OP-CEC2	OP-CEC2
13. P-OPERATION-CRE-DEST-CGER	OP-CRE2	OP-CRE2

1.7. Ajout des conditions et actions primitives

Pour achever de construire P1, il reste à ajouter les conditions et actions primitives. Nous nous baserons pour ce faire sur les spécifications des différents composants.

1.7.1. Spécifications des différents composants

Pour éviter de spécifier les composants un par un, nous donnerons ci-dessous une spécification générale valable pour les composants n° 1 à 9. Certains adaptations seront parfois nécessaires, nous les mentionnerons ci-après.

Spécification générale des composants correspondance (n° 1 à 9)

Soit un composant correspondance exprimant une correspondance entre un composant de FICH-IN (d'occurrence notée $C_{FICH-IN}$) et un composant de F2 (d'occurrence notée C_{F2}); sa spécification est la suivante :
lit un $C_{FICH-IN}$ et génère un C_{F2} moyennant les conventions suivantes:
le premier enregistrement de $C_{FICH-IN}$ a déjà été lu, après l'exécution de ce composant, l'enregistrement suivant le dernier de $C_{FICH-IN}$ sera lu s'il existe, sinon la fin de fichier aura été déclenchée.

Cette spécification est parfois à adapter ou à compléter comme suit :
- La spécification du composant PROCESS-FICH-IN est équivalente à la spécification générale d'un composant correspondance à ceci près que le premier enregistrement de FICH-IN n'a pas encore été lu.

- La spécification du composant P-GROUPE-CREANCIER-CEC est à compléter par : reçoit en plus en W.S.S. le numéro d'identification de l'enregistrement DEBUT-CEC, la date pivot et la provenance des descriptions d'opérations du groupe créancier CEC.

- La spécification du composant P-GROUPE-CONTRAT-CRE (P-GROUPE-CONTRAT-CEC) est à compléter par :
reçoit en plus en W.S.S. le numéro d'identification de l'enregistrement DEBUT-CRE (DEBUT-CEC), la date pivot, la provenance et le numéro d'identification du créancier des descriptions d'opérations du groupe contrat-cré (GROUPE CONTRAT-CEC).

Mêmes modifications aux spécifications des composants
P-GPE-CONTRAT-CEC-DEST-CEC, P-GPE-CONTRAT-CEC-DEST-CGER,
P-GPE-CONTRAT-CRE-DEST-CEC, P-CPE-CONTRAT-CRE-DEST-CGER en supprimant ce qu'il y a entre parenthèses et en remplaçant DEBUT-CRE et groupe contrat cré respectivement par
DEBUT-CEC , groupe contrat cec destiné cec ; DEBUT-CEC, groupe contrat cec destiné CGER
DEBUT CRE, groupe contrat cré destiné cec ; DEBUT-CRE, groupe contrat cré destiné CGER

Spécification du composant P-OPERATION-CRE-DEST-CGER (P-OPERATION-CEC-DEST-CGER)

- génère un enregistrement $OP\text{-}CRE_2$ ($OP\text{-}CEC_2$) correspondant au dernier enregistrement lu $OP\text{-}CRE_2$ ($OP\text{-}CEC_2$) de FICH-IN ;
- lit l'enregistrement suivant s'il existe, sinon déclenche la fin de fichier.

Spécification de P-OPERATION-CEC-DEST-CEC (P-OPERATION-CRE-DEST-CEC)

- lit l'enregistrement suivant s'il existe, sinon déclenche la fin de fichier.

Spécification de PROCESS-DEBUT-CEC (PROCESS-DEBUT-CRE)

- mémorise en W.S.S. le numéro d'identification de l'enregistrement début-cec (début-cré), la date pivot, la provenance (et le numéro d'identification du créancier) des descriptions d'opérations du F.L cec (créancier) courant ;
- lit l'enregistrement suivant.

Spécification de PROCESS-FIN-CEC (PROCESS-FIN-CRE)

- le dernier enregistrement lu est fin-cec (fin-cré) ;
- lit l'enregistrement suivant s'il existe, sinon déclenche la fin de fichier.

Spécification de P-ENTETES-CEC (CRE)

- génère un enregistrement entête-créancier tel défini au paragraphe 1.4 si le groupe contrat cec destiné à la CGER est le premier groupe contrat cec (cre) destiné à la CGER du groupe cré-cec (du groupe cré) courant, sinon ne génère rien.

Spécification de P-ENTETE-CONTRAT-F.L CEC (FL CRE)

- Génère un enregistrement entête-contrat tel défini au paragraphe 1.4.

Spécification de P-ENT-CREANCIER-CEC (CRE)

- génère un enregistrement entête-créancier tel défini au paragraphe 1.4.

Concernant les composant P-BODY..., il ne nous a pas semblé nécessaire d'en écrire explicitement les spécifications, car elles se déduisent aisément des composants inférieurs ou supérieurs.

Nous ne spécifierons pas ici les composants P-DEBUT-CREANCIER-CEC, P-DEBUT-GPE-CONTRAT-CEC-DEST-CEC, P-DEBUT-GPE-CONTRAT-CRE-CGER, P-DEBUT-GPE-CONTRAT-CRE-DEST-CEC, P-DEBUT-GPE-CONTRAT-CRE-DEST-CGER, car ils correspondent à des instructions d'initialisation ou de clôture qui seront déterminées lorsqu'on programmera les composants dont ils dépendent.

1.7.2. Détermination des conditions et actions primitives

1.7.2.1. Détermination des conditions d'itération

Pour déterminer les conditions d'itération on se réfèrera aux spécifications des composants itération.

- Détermination de C1 : le sous-composant itération P-GROUPE doit être itéré autant de fois qu'il y a de fichiers logiques dans FICH-IN puisque PROCESS-FICH-IN lit le fichier FICH-IN. L'itération se terminera lorsqu'il n'y aura plus de fichier logique à traiter, c'est-à-dire lorsqu'on sera en fin de fichier. Pour détecter cet état, nous utiliserons une variable EOF-FICH-IN qui sera initialisée à 0 et mise à 1 lors d'une opération de lecture qui déclenche la fin du fichier. La condition C1 se traduira donc dans le texte cobol par UNTIL EOF-FICH IN = 1.

- Détermination de C4 : le sous-composant itération P-GROUPE-CREANCIER-CEC doit être itéré autant de fois qu'il y a de groupes créanciers cec dans un groupe cec, c'est-à-dire jusqu'à ce que l'on rencontre un enregistrement FIN-CEC. Cet enregistrement est caractérisé par un n° d'identification (ID-ENREG) ayant la valeur 9. La condition C4 devient donc en cobol UNTIL ID-ENREG = 9.

- Détermination de C5 : le sous-composant itération P-GROUPE-CONTRAT-CEC doit être itéré autant de fois qu'il y a de groupes contrats cec dans un groupe créancier cec, c'est-à-dire jusqu'à ce que l'on rencontre un enregistrement FIN-CEC ou un nouveau groupe créancier cec. Un nouveau groupe créancier cec débutera dès qu'on aura lu en enregistrement OP-CEC dont le numéro d'identification de créancier diffère de celui du groupe créancier cec courant. Ce dernier numéro est mémorisé dans une zone en W.S.S. (NU-ID-CRE OF W-ENTETE-CREANCIER). L'enregistrement FIN-CEC est caractérisé par un numéro d'identification (ID-ENREG) ayant la valeur 9. La condition C5 devient donc en cobol : UNTIL ID-ENREG = 9 OR
NU-ID-CRE OF OP-CEC NOT =
NU-ID-CRE OF W-ENTETE-CREANCIER

- Détermination de C6 : le sous-composant itération P-GROUPE-CONTRAT-CRE doit être itéré autant de fois qu'il y a de groupes contrats cré dans un même groupe créancier, c'est-à-dire jusqu'à ce que l'on rencontre l'enregistrement FIN-CRE. Cet enregistrement est caractérisé par un numéro d'identification (ID-ENREG) ayant la valeur 9. La condition C6 se traduit donc en cobol par UNTIL ID-ENREG = 9.

- Détermination de C11, C12 : les sous-composants itérations P-OPERATION-CEC-DEST-CEC, P-OPERATION-CEC-DEST-CGER, sont itérés autant de fois qu'il y a d'opérations respectivement dans un groupe contrat cec destiné cec, dans un groupe contrat cec destiné CGER, l'itération se terminera soit lorsqu'on aura rencontré un enregistrement FIN-CEC, c'est-à-dire lorsqu'on aura rencontré un enregistrement dont le numéro d'identification = 9, soit lorsqu'on sera arrivé en fin de groupe créancier cec (cette condition est équivalente à C5), soit lorsqu'on sera arrivé en fin de groupe contrat cec, c'est-à-dire lorsqu'on aura lu un enregistrement OP-CEC dont le numéro de domiciliation diffère de celui du groupe contrat cec courant (mémorisé dans une zone en WSS NU-DOMI OF W-ENTETE-CONTRAT). Les conditions C11 et C12 se traduisent dans le texte cobol par :
UNTIL ID-ENREG = 9 OR NU-ID-CRE OF OP-CEC NOT = NU-ID-CRE OF
W-ENTETE-CREANCIER OR NU-DOMI OF OP-CEC NOT = NU-DOMI OF
W-ENTETE-CONTRAT.

- Détermination de C13, C14 : les sous-composants itérations P-OPERATION-CRE-DEST-CEC, P-OPERATION-CRE-DEST-CGER sont itérés autant de fois qu'il y a d'opérations respectivement dans un groupe contrat cré destiné cec, dans un groupe contrat cré destiné CGER. L'itération se terminera soit lorsqu'on aura rencontré un enregistrement FIN-CRE, soit lorsqu'on sera arrivé en fin de groupe contrat cré, c'est-à-dire lorsqu'on aura lu un enregistrement OP-CRE dont le numéro de domiciliation diffère de clui du groupe contrat cré courant (mémorisé dans la zone NU-DOMI OF W-ENTETE-CONTRAT en WSS).
Les conditions C11 et C12 deviennent
UNTIL ID-ENREG = 9 OR NU-DOMI OF OP-CRE NOT = NU-DOMI OF
W-ENTETE-CONTRAT

1.7.2.2. Détermination des conditions de sélection

- Détermination de C2 : le composant PGROUPE est un composant P-GROUPE-CEC si le fichier logique à traiter est originaire du CEC, c'est-à-dire si la provenance de l'ensemble des descriptions d'opérations de ce fichier logique est égal au numéro d'identification de CEC (102).
Cette condition se traduit en cobol par IF PROVENANCE OF DEBUT-CEC = 102.

- La condition C3 est la négation de C2.

- Détermination de C8 (C10) : le composant P-GROUPE-CONTRAT-CEC (P-GROUPE-CONTRAT-CRE) est un composant P-GPE-CONTRAT-CEC-DEST-CGER (P-GROUPE-CONTRAT-CRE-DEST-CGER) si le groupe contrat CEC (le groupe contrat cré) est destiné à la CGER, c'est-à-dire si le numéro de domiciliation du contrat de ce groupe commence par 048. Cette condition se traduira en cobol comme suit :
IF DEBUT-NU-DOMI OF OP-CEC (OP-CRE) = 048.

- Les conditions C7 et C9 sont respectivement la négation de C8, C10.

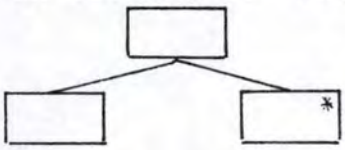
- Détermination de C15 (C17) : le composant P-ENTETES-CEC (P-ENTETES-CRE) est un composant P-ENT-CRANCIER-CEC (P-ENT-CREANCIER-CRE) si le groupe contrat courant est le premier groupe contrat destiné à la CGER du groupe créancier courant. Nous utiliserons,

pour ce faire, une variable ENTETE-IMPRIMEE initialisée à 0 au début d'un nouveau groupe CRE CEC (groupe CRE) et mise à 1 dès que l'on écrit l'entête - créancier correspondant à ce groupe cre cec (groupe cre) (c'est-à-dire dès que l'on rencontre le premier groupe contrat destiné à la CGER).

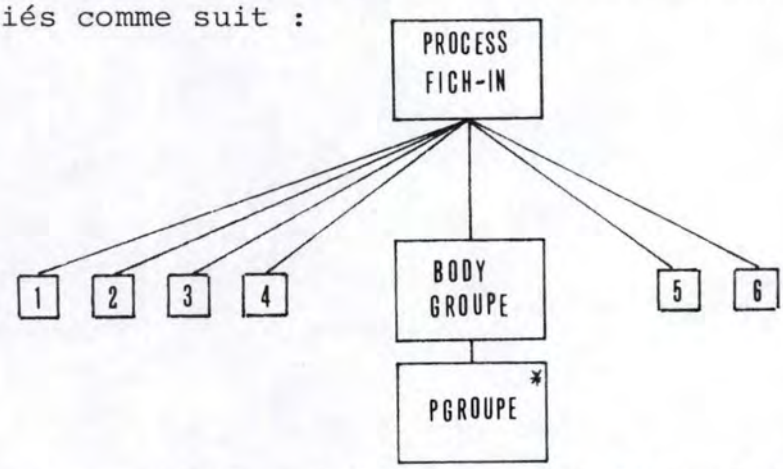
Les conditions C16 et C18 sont respectivement la négation de C15, C17.

1.7.2.3. Détermination des actions primitives

Quand on examine la spécification de PROCESS-FICH-IN, on constate qu'elle n'est pas équivalente à une simple itération du composant P-GROUPE. En effet, avant la première itération de P-GROUPE, on aura dû (1) ouvrir le fichier FICH-IN en lecture, (2) ouvrir le fichier F2 en écriture, (3) avoir nié la condition C1, (4) avoir lu le permier enregistrement. De plus après avoir traité le dernier fichier logique on aura dû : (5) fermer les fichiers, (6) arrêter l'exécution.
Nous ajouterons ces actions primitives en les représentant par leur numéro inscrit dans un rectangle. Les structures mixtes telles :



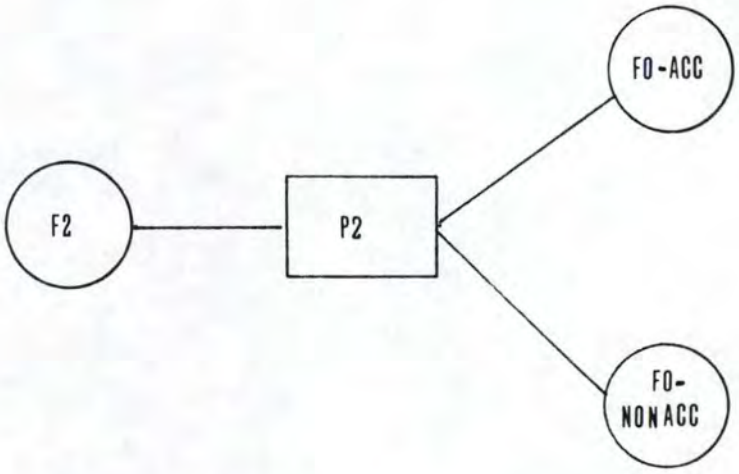
n'étant pas admises, nous sommes obligés d'ajouter un niveau supplémentaire dans l'arbre. Les deux premiers niveaux seront modifiés comme suit :



Les autres actions primitives seront déterminées lorsqu'on programmera les différents composants.
Le programme se trouve à l'annexe 2.

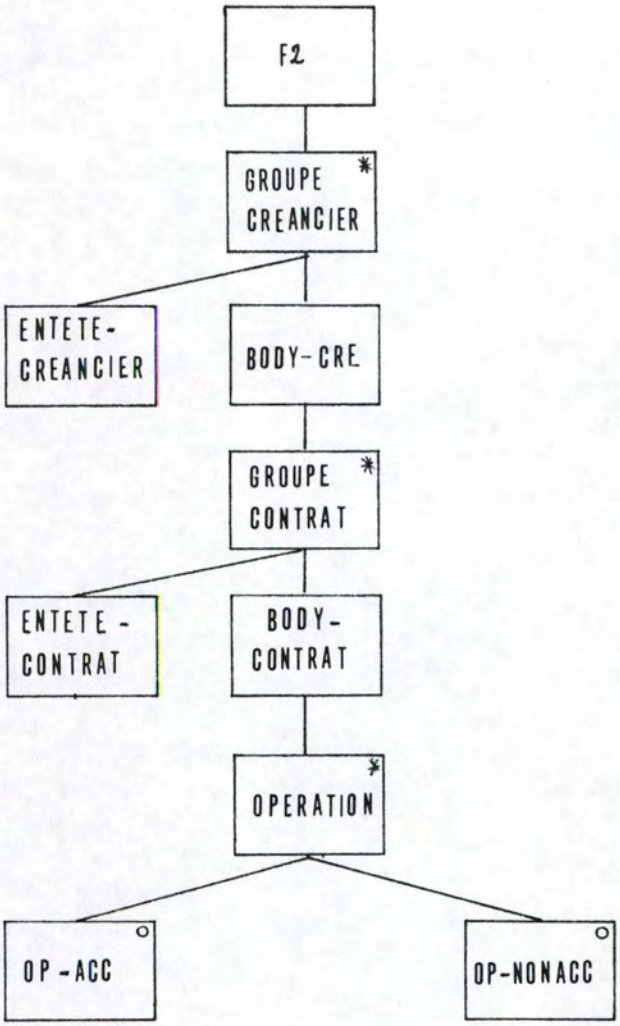
1.8. Spécification de P2

- reçoit en entrée un ensemble de descriptions d'opérations au sens strict représenté par le fichier F2 selon les conventions mentionnées au paragraphe 1.4. ;
- génère en sortie deux sous-ensembles de descriptions d'opération :
  * un sous-ensemble de descriptions d'opérations acceptables représenté par le fichier FO-ACC ; chacune de ces descriptions d'opérations acceptables est représentée par un enregistrement ENREG-ACC ; les conventions de représentation de ces descriptions sont évidentes ;
  * un sous-ensemble de descriptions d'opérations non acceptables représenté par le fichier FO-NONACC ; chacune de ces descriptions est représentée par un enregistrement ENREG-NONACC ; les conventions de représentation de ces descriptions sont également évidentes.



1.9. Structuration des entrées et sorties de P2

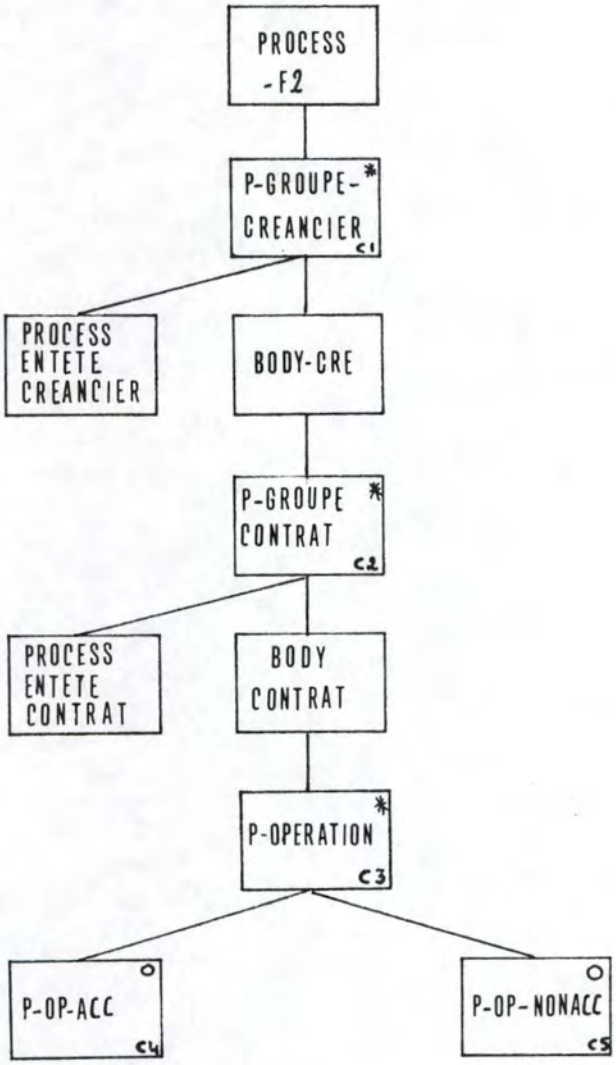
La structure des entrées est une structure pouvant être placée sur F2 et être mise en correspondance avec la structure des sorties. Cette structure est celle décrite au paragraphe 1.4. graphiquement elle se représente comme suit :



La sortie est constituée de 2 flux structurés très simplement : ils sont tous les 2 vus comme une suite d'enregistrements.

1.10. Recherche des correspondances

Nous ne nous trouvons pas ici en présence d'un cas à un flux d'entrée et à un flux de sortie. Pour la recherche des correspondances, nous construirons une structure de données hypothétique correspondant à la fusion des différentes sorties. Ainsi sur base de la structure hypothétique des sorties et la structure des entrées, nous serons amenés à construire l'arbre fusion suivant :



1.11. Ajout des conditions et actions primitives

Nous procéderons comme pour le programme P1 : nous spécifierons d'abord les différents composants de l'arbre fusion ensuite, sur base de ces spécifications, nous déterminerons les conditions et actions primitives à ajouter.

1.11.1. Spécifications des différents composants

- Définition

* Un groupe créancier est un enregistrement entête-créancier suivi d'un ensemble de groupe contrat.
* Un groupe contrat est un enregistrement entête-contrat suivi d'un ensemble d'enregistrements opération.

- La spécification du composant PROCESS F2 correspond à la spécification du programme P2.

- Spécification de P-GROUPE-CREANCIER

* lit un groupe créancier dont le premier enregistrement (entête-créancier) a déjà été lu ;
* génère pour chaque enregistrement opération de ce groupe créancier un enregistrement ENREG-ACC si la description de l'opération (représentée par l'enregistrement OPERATION dans le groupe créancier) a satisfait aux 7 contrôles indiqués au paragraphe 3.2. du chapitre 1 sinon il génère un enregistrement ENREG-NONACC ;
* lit l'enregistrement suivant le dernier du groupe créancier s'il existe, sinon déclenche la fin de fichier.

- Spécification de P-GROUPE-CONTRAT

* lit un groupe contrat dont le premier enregistrement a déjà été lu (ce groupe contrat fait partie d'un groupe créancier X) ;
* reçoit en WSS les informations communes à tous les groupes contrat du groupe créancier X à savoir la date pivot, la provenance, le numéro d'identification du créancier ;
* reçoit le résultat des 2 premiers contrôles avec la description de l'opposition sur créancier si elle existe ;
* génère pour chaque enregistrement OPERATION de ce groupe contrat

un enregistrement ENRE-ACC si la description de l'opération (représentée par l'enregistrement OPERATION dans le groupe contrat) a satisfait aux 7 contrôles cités au paragraphe 3.2. du chapitre 1, sinon il génère un enregistrement ENREG-NONACC.

* lit l'enregistrement suivant le dernier du groupe contrat s'il existe, sinon déclenche la fin de fichier.

- Spécification de P-OPERATION

* le dernier enregistrement lu est un enregistrement OPERATION faisant partie d'un groupe contrat lui-même inclus dans un groupe créancier ;
* reçoit un WSS les informations communes à tous les enregistrements de ce groupe contrat dans ce groupe créancier à savoir la date pivot, la provenance, le numéro d'identification du créancier et le numéro de domiciliation ;
* reçoit en WSS le résultat des 6 premiers contrôles avec les descriptions d'opposition sur créancier et sur contrat si elles existent, la date de révocation si elle existe, le numéro d'identification de créancier se trouvant sur le contrat de domiciliation si il est compatible avec celui de la description d'opération courante, le numéro de compte débiteur s'il existe, et l'identification de la gestion si elle peut être déduite ;
* génère un enregistrement ENREG-ACC si la description de l'opération (représentée par l'enregistrement OPERATION) a satisfait aux 7 contrôles mentionnés au paragraphe 3.2. du chapitre 1, sinon génère un enregistrement ENREG-NONACC.

- Spécification de PROCESS-ENTETE-CREANCIER

* le dernier enregistrement lu est un enregistrement ENTETE-CREANCIER ;
* réalise les contrôles 1 et 2 ;
* mémorise les résultats de ces contrôles en WSS ainsi que la description de l'opposition sur créancier si elle existe ;
* lit l'enregistrement suivant s'il existe sinon déclenche la fin de fichier.

- Spécification de PROCESS-ENTETE-CONTRAT

* le dernier enregistrement lu est un enregistrement ENTETE-CONTRAT ;
* réalise les contrôles 3, 4, 5, 6 ;
* mémorise les résultats de ces contrôles en WSS ainsi que le n° de compte débiteur, l'identification de la gestion, la description de l'opposition sur contrat et la date de révocation s'ils existent ;
* lit l'enregistrement suivant s'il existe sinon déclenche la fin de fichier.

Les spécifications des composants BODY... ainsi que celles des composants P-OP-ACC, P-OP-NONACC sont évidentes.

1.11.2. Détermination des conditions et actions primitives

1.11.2.1. Détermination des conditions d'itération

- Détermination de C1 : le sous-composant itération P-GROUPE-CREANCIER doit être itéré autant de fois qu'il y a de groupes créancier dans F2. L'itération se terminera lorsqu'on sera arrivé en fin de fichier, nous utiliserons pour détecter cet état une variable EOF-F2 initialisée à 0 et mise à 1, lors d'une opération de lecture qui déclenche la fin de fichier. La condition C1 devient en cobol : UNTIL EOF-F2 = 1.

- Détermination de C2 : Le sous-composant itération P-GROUPE-CONTRAT est itéré autant de fois qu'il y a de groupes contrat dans un groupe créancier. L'itération s'achève donc lorsqu'on arrive en fin de fichier ou lorsqu'on rencontre un nouveau groupe créancier. Sachant que le premier enregistrement d'un groupe créancier est un enregistrement ENTETE-CREANCIER caractérisé par un n° d'identification (ID-ENREG) ayant la valeur 0 et que la fin de fichier est détectée grâce à la valeur de la variable EOF-F2, la condition C2 devient en cobol : UNTIL EOF-F2 = 1 OR ID-ENREG OF ENREG-F2 = 0.

- Détermination de C3 : le sous-composant itération P-OPERATION doit être itéré autant de fois qu'il y a d'enregistrements OPERATION dans un groupe contrat. L'itération se terminera soit lorsqu'on arrive en fin de fichier, soit lorsqu'on rencontre un nouveau groupe créancier, soit lorsqu'on rencontre un nouveau groupe contrat.

Un groupe contrat débute par un enregistrement ENTETE-CONTRAT caractérisé par un n° d'identification ayant la valeur 2.
La condition C3 devient : UNTIL EOF-F2 =1 OR
ID-ENREG OF ENREG-F2 = 0 OR ID-ENREG OF ENREG-F2 = 2

1.11.2.2. Détermination des conditions de sélection

- Détermination de C4 : le composant P-OPERATION est un composant P-OP-ACC si la description de l'opération courant a satisfait aux 7 contrôles (du paragraphe 3.2. chapitre 1) sinon c'est un composant P-OP-NONACC.
Pour traduire facilement cette condition, les résultats des 7 contrôles sont regroupés, ils constituent les éléments d'un tableau TAB mémorisé en WSS. Si la description de l'opération a satisfait aux 7 contrôles, chaque zone mémorisant le résultat d'un des 7 contrôles doit avoir été mise à zéro, le contenu du tableau TAB doit donc être égal à une suite de 7 zéros ; ce qui se traduira en cobol par IF TAB OF W-ENREG = ALL ZERO

- La condition C5 est la négation de C4.

1.11.2.3. Détermination des actions primitives

Même raisonnement que pour P1 en remplaçant PROCESS-FICH-IN par PROCESS-F2, P-GROUPE par P-GROUPE-CREANCIER, F2 par FO-ACC, FO-NONACC et ensuite FICH-IN par F2.

Le programme se trouve à l'annexe 3.

1.12. Inversion

Après avoir construit P1 et P2, nous avons supprimé le fichier intermédiaire F2 en appliquant les règles d'inversion expliquées dans la première partie de ce travail.
En inversant, nous pouvions soit faire de P1 un sous-programme, soit faire de P2 un sous-programme. Nous avons choisi la première solution.
Les programmes obtenus de la sorte se trouvent à l'annexe 4.

2. CONSTRUCTION SELON L'APPROCHE "EXPLICITATION DES RAISONNEMENTS"

2.1. Remarques

La partie spécification présentée au chapitre 1 reste valable à ceci près : la notion de description complétée d'opération a été redéfinie autrement ; de plus,en sortie nous avons décidé ici de faire apparaître à la fois dans l'enregistrement ENREG-ACC et ENREG-NONACC l'origine de la description de l'opération.

2.2. Descriptions des zones Cobol

Voir annexe 5 - partie DATA DIVISION.

2.3. Définitions

Un groupe CEC est un fichier logique (originaire du) CEC.

Un groupe CREANCIER est un fichier logique (originaire d'un) CREANCIER.

Un groupe CREANCIER-CEC est une suite non vide, de taille maximale d'enregistrements OP-CEC consécutifs d'un même fichier logique CEC et ayant même numéro d'identification de créancier.

Un groupe CONTRAT CEC est une suite non vide, de taille maximale d'enregistrements OP-CEC consécutifs d'un même fichier logique CEC et ayant même numéro de domiciliation.

On distingue 2 types de groupes contrat-cec :
les groupes CONTRAT-CEC-DEST-CEC : ceux dont le numéro de domiciliation n'est pas celui d'un contrat pour lequel la CGER est l'institution domicile (i.e : le numéro de domiciliation ne commence pas par 048) ;

les groupes CONTRAT-CEC-DEST-CGER : les autres.

Les notions de groupe CONTRAT-CRE, groupe CONTRAT-CRE-DEST-CEC, groupe CONTRAT-CRE-DEST-CGER sont définies de la même façon en remplaçant "fichier logique CEC" et OP-CEC, respectivement par "fichier logique CREANCIER" et "OP-CRE".

FIN DE FICHIER FICH-IN : on introduit une variable EOF-FICH-IN définie ssi on a au moins exécuté une instruction de lecture relative à ce fichier. Sa valeur sera déterminée à un instant donné I, par l'effet de l'exécution de la dernière instruction de lecture précédant cet instant I.
Elle sera égale à 0 si l'exécution de cette dernière instruction a eu pour effet de placer dans le buffer un nouvel article du fichier FICH-IN, sinon elle sera égale à 1.

DEBUT D'UN GROUPE : on dit par définition qu'on est au début d'un groupe ssi le buffer contient le premier enregistrement de ce groupe.

FIN D'UN GROUPE : on dit qu'on est à la fin d'un groupe quand le contenu du buffer est égal au premier enregistrement suivant le dernier du groupe si il existe ou qu'on est en fin de fichier si le dernier enregistrement du groupe est le dernier du fichier.

TRAITEMENT D'UN GROUPE : on dit, par définition, qu'un programme traite un groupe si avant son exécution on est au début de ce groupe et qu'après on est à la fin de ce groupe.

GENERATION RELATIVE A UN GROUPE : on dit, par définition qu'un programme effectue la génération relative à un groupe ssi après son exécution on a généré dans les fichiers de sortie (FO-ACC, FO-NONACC) les descriptions complétées des opérations de ce groupe (destinées à la CGER).

Pour un groupe (différent d'un groupe CEC ou CREANCIER) on définit la notion d'ENSEMBLE DE DESCRIPTIONS D'OPERATIONS REPRESENTE PAR UN GROUPE DANS SON FICHIER LOGIQUE.
Un groupe est une suite d'enregistrements OP (-CEC ou -CRE).
A chaque enregistrement OP (-CEC ou -CRE) correspond une et une seule description d'opération.
Cette description d'opération est constituée d'informations provenant à la fois de l'enregistrement OP et de l'enregistrement DEBUT (*)

(*) pour la répartition exacte des informations (constituant une description) entre enregistrement DEBUT et OPERATION se référer au paragraphe 3.4 du chapitre 1.

Cet enregistrement DEBUT ne fait pas partie du groupe, mais bien du fichier logique contenant le groupe.
C'est pourquoi on ne parlera pas d'ensemble de descriptions d'opérations représenté par un groupe mais bien d'ensemble de descriptions d'opérations représenté par un groupe dans son fichier logique.

L'ENTETE D'UN GROUPE EST COURANTE si les valeurs des zones DATE-PIVOT, PROVENANCE et ORIGINE en W.S.S. sont respectivement la date pivot, la provenance et l'origine de l'ensemble des descriptions d'opérations représenté par ce groupe dans le fichier logique auquel il appartient.

UN PROGRAMME NE MODIFIE PAS L'ENTETE COURANTE si au début de son exécution l'entête d'un groupe est courante et qu'à la fin l'entête de ce même groupe est encore courante.

PROPRIETE : de la définition "l'entête d'un groupe est courante" on déduit qu'un programme ne modifie pas l'entête courante si il ne modifie pas les valeurs des zones DATE-PIVOT, PROVENANCE, _ORIGINE EN W.S.S., c'est-à-dire la zone D'ENTETE.

LE NU-ID-CRE D'UN GROUPE EST COURANT SI la valeur de la zone NU-ID-CRE en W.S.S. est le numéro d'identification du créancier de l'ensemble des descriptions d'opérations représenté par ce groupe dans son fichier logique. (Rmq : du créancier signifie qu'il doit être unique, ceci implique que le groupe dont on parle ici doit être ≠ d'un groupe cec).

UN PROGRAMME NE MODIFIE PAS LE NU-ID-CRE courant si au début de son exécution le nu-id-cré d'un groupe est courant et qu'en fin d'exécution le nu-id-cré de ce même groupe est encore courant.

PROPRIETE : un programme ne modifie pas le nu-id-cré courant si il ne modifie pas la valeur de la zone nu-id-cré en W.S.S.

On définit de la même façon :

1. les notions de NU-CPTE-CRE courant d'un groupe,de programme ne modifiant pas le nu-cpte-cré courant et la propriété qui en découle en remplaçant respectivement "NU-ID-CRE" et "huméro d'identification de créancier" par "NU-CPTE-CRE" et "numéro de compte créancier".

2. les notions de NU-DOMI courant d'un groupe, de programme ne modifiant pas le nu-domi courant et la propriété qui en découle en remplaçant respectivement "NU-ID-CRE" et "numéro d'identification de créancier" par "NU-DOMI" et "numéro de domiciliation".

Le nu-id-cré d'un groupe est vérifié ssi il est courant et
- la zone ETAT-CRE en W.S.S. contient la valeur 1 (traduisant le fait que le créancier identifié par le numéro d'identification nu-id-cré est vérifié),
- la zone EXISTE-CRE en W.S.S. contient la valeur 1 si ce créancier existe, 0 sinon ;
- si ce créancier existe, la zone OPPOSITION-CRE contient la valeur 1 si il y a une opposition sur ce créancier, 0 sinon ;
- si ce créancier existe et si il y a une opposition sur ce créancier, la zone DESC-OPPOSITION-CRE contient la description de l'opposition sur ce créancier.

Le NU-ID-CRE d'un groupe est non vérifié ssi il est courant et que la valeur de la zone ETAT-CRE est égale à 0.

Un programme ne modifie pas le nu-id-cré vérifié ssi au début de son exécution le nu-id-cré d'un groupe est vérifié et qu'à la fin de l'exécution le nu-id-cré de ce même groupe est encore vérifié.

Un programme ne modifie pas le nu-id-cré non vérifié ssi au début de son exécution le nu-id-cré d'un groupe est non vérifié et qu'à la fin de l'exécution le nu-id-cré de ce même groupe est encore non vérifié.

PROPRIETE : un programme ne modifie pas la nu-id-cré vérifié (ou non vérifié) s'il ne modifie pas la valeur de la zone DCREANCIER en W.S.S., c'est-à-dire la zone reprenant toutes les zones citées dans la définition de la propriété "le nu-id-cré d'un groupe est vérifié".

Le NU-DOMI d'un groupe est vérifié : ssi il est courant et
* la zone EXISTE-CDD contient la valeur 1 si le contrat de domiciliation identifié par le numéro de domiciliation NU-DOMI existe, 0 sinon.

* si le contrat de domiciliation identifié par le numéro de domiciliation existe

alors - la zone NU-ID-CRE contient le numéro d'identification du créancier repris dans ce contrat,
- la zone NU-REF-CDD contient le numéro de référence repris dans ce contrat,
- la zone OPPOSITION-CDD contient la valeur 1 s'il y a une opposition sur ce contrat, 0 sinon,
- la zone ETAT-CDD contient la valeur 1 si ce contrat est révoqué, 0 s'il est en cours,

si ce contrat est révoqué alors
la zone DATE-REVOCATION contient la date de révocation ;

si il y a une opposition sur ce contrat alors
la zone DESC-OPPOSITION-CDD contient la description de l'opposition sur ce contrat ;

- la zone NU-CPTE-DEBITEUR contient le numéro de compte du débiteur repris dans ce contrat ;
- la zone ID-GESTION contient l'identification de la gestion vers laquelle l'opération est destinée,

sinon la zone NU-CPTE-DEBITEUR est mise à blanc ainsi que la zone ID-GESTION.

Un programme ne modifie pas le nu-domi vérifié :
ssi au début de son exécution le nu-domi d'un groupe est vérifié et qu'à la fin le nu-domi de ce même groupe est encore vérifié.

PROPRIETE : un programme ne modifie pas le nu-domi vérifié ssi il ne modifie pas la valeur de la zone DCONTRAT reprenant toutes les zones mentionnées dans la définition de la propriété "le nu-domi d'un groupe est vérifié".

Une description réduite d'opération est courante ssi
- le nu-cpte-cré de la description d'opération (*) est courant
- la zone nature-OP OF DOPERATION contient la nature de la description d'opération,
- la zone montant-OP OF DOPERATION contient le montant de la description d'opération,
- la zone nom-cré OF DOPERATION contient le nom du créancier de la description d'opération,

(*) représenté par l'enregistrement courant dans son fichier logique.

- les zones COMM1 ,COMM2 contiennent les communications de la description d'opération,
- la zone NU-REFERENCE contient le numéro de référence de la description d'opération.

Une description complétée d'opération est courante ssi
- la description réduite de cette opération est courante,
- le nu-id-cré de la description d'opération est vérifié,
- le nu-domi " " " " est vérifié,
- l'entête " " " " est vérifié,
- la liste des résultats des 7 contrôles "réalisés" pour cette description d'opération est représentée par un tableau TAB de 7 éléments ELE tel que
  * ELE OF TAB(1) = 0 si le nu-id-cré existe
                 1 sinon
  * si le nu-id-cré existe
    alors
        ELE OF TAB(2) = 0 si il n'y a pas d'opposition sur le créancier identifié par nu-id-cré
                      1 sinon
    sinon le contenu de ELE OF TAB (2) est indéterminé.
  * ELE OF TAB (3) = 0 si le nu-domi existe
                 1 sinon
  * si le nu-domi existe
    alors
        ELE OF TAB(4) = 0 si le contrat de domiciliation identifié par nu-domi est en cours
                      1 sinon
        ELE OF TAB(5) = 0 si il n'y a pas d'opposition sur ce contrat
                      1 sinon
        ELE OF TAB(6) = 0 si il y a compatibilité entre le numéro d'identification du créancier mentionné dans la description de cette opération et celui repris dans le contrat de domiciliation identifié par nu-domi
                      1 sinon
        ELE OF TAB(7) = 0 si il y a compatibilité entre le numéro de référence mentionné dans

la description de cette opération et
celui repris dans le contrat de domici-
liation identifié par nu-domi
1 sinon

sinon le contenu des zones ELE OF TAB(i) pour 4 ⩽ i ⩽ 7 est indéterminé.

2.4. Spécifications des sections

2.4.1. P-GROUPE-CEC

Précondition : on est au début d'un groupe cec.

Effet : traite ce groupe cec
effectue la génération relative à ce groupe cec.

2.4.2. P-GROUPE-CRE

Précondition : on est au début d'un groupe créancier.

Effet : traite ce groupe créancier,
effectue la génération relative à ce groupe créancier.

2.4.3. P-GROUPE-CREANCIER-CEC

Précondition : on est au début d'un groupe créancier-cec,
l'entête de ce groupe est courante.

Effet : traite ce groupe créancier,
effectue la génération relative à ce groupe créancier-cec,
ne modifie pas "l'entête courante".

2.4.4. P-GROUPE-CONTRAT-CEC-DEST-CEC

Précondition : on est au début d'un groupe contrat-cec-dest-cec,
l'entête de ce groupe est courante,
le nu-id-cré de ce groupe est vérifié ou non vérifié,

Effet : traite ce groupe contrat-cec-dest-cec,
n'effectue aucune génération,
ne modifie ni l'entête courante ni le nu-id-cré vérifié (ou non vérifie).

2.4.5. P-GROUPE-CONTRAT-CEC-DEST-CGER

Précondition : on est au début d'un groupe contrat-cec-dest-cger,
l'entête de ce groupe est courante,
le nu-id-cré de ce groupe est vérifié.

Effet : traite ce groupe contrat-cec-dest-cger,
effectue la génération relative à ce groupe,
ne modifie ni l'entête courante, ni le nu-id-cré vérifié.

2.4.6. P-GROUPE-CONTRAT-CRE-DEST-CEC

Précondition : on est au début d'un groupe contrat-cré-dest-cec,
l'entête de ce groupe est courante,
le nu-id-cré de ce groupe est vérifié ou non vérifié,
le nu-cpte-cré de ce groupe est courant.

Effet : traite ce groupe contrat-cré-dest-cec,
n'effectue aucune génération,
ne modifie ni l'entête courante, ni le nu-id-cré vérifié ou non vérifié ni le nu-cpte-cré courant.

2.4.7. P-GROUPE-CONTRAT-CRE-DEST-CGER

Précondition : on est au début d'un groupe contrat-cré-dest-cger,
l'entête de ce groupe est courante,
le nu-id-cré de ce groupe est vérifié,
le nu-cpte-cré de ce groupe est courant,

Effet : traite ce groupe contrat-cré-dest-cger,
effectue la génération relative à ce groupe,
ne modifie ni l'entête courante, ni le nu-id-cré vérifié ni le nu-cpte-cré courant.

2.4.8. VERIFICATION-NU-ID-CRE

Précondition : le nu-id-cré d'un groupe est courant,

Postcondition : le nu-id-cré de ce groupe est vérifié,

Effet : ne modifie que le contenu de la zone DCREANCIER en W.S.S.
laisse inchangé l'état des fichiers.

2.4.9. VERIFICATION-NU-DOMI

Précondition : le nu-domi d'un groupe est courant.

Postcondition : le nu-domi de ce groupe est vérifié.

Effet : ne modifie que le contenu de la zone DCONTRAT (en W.S.S.),
laisse inchangé l'état des fichiers.

2.4.10. CALCUL D'UNE DESCRIPTION-COMPLETEE-CEC

Précondition : le buffer contient un enregistrement OP-CEC,
l'entête de cet enregistrement est courante,
le nu-id-cré de cet enregistrement est vérifié,
le nu-domi de cet enregistrement est vérifié.

Postcondition : la description complétée de cet enregistrement est courante,
l'état des fichiers est non modifié.

2.4.11. CALCUL D'UNE DESCRIPTION-COMPLETEE-CRE

Précondition : le buffer contient un enregistrement OP-CRE
l'entête de cet enregistrement est courante,
le nu-id-cré de cet enregistrement est vérifié,
le nu-cpte-cré de cet enregistrement est courant,
le nu-domi de cet enregistrement est vérifié.

Postcondition : la description complétée de cet enregistrement est courante,
l'état des fichiers est non modifié.

2.4.12. CREEXI

Reçoit la description d'un créancier (DCREANCIER) réduite au numéro d'identification de ce créancier (IDENT-CREANCIER) ; renvoie la description complète de ce créancier, c'est-à-dire une variable EXISTE-CRE ayant la valeur 1 s'il existe un créancier identifié par ce numéro d'identification de créancier, 0 sinon.
Dans le cas où EXISTE-CRE = 1, il renverra dans la variable OPPOSITION-CRE la valeur 1 s'il y a opposition sur ce créancier, la variable DESC-OPPOSITION-CRE contiendra la description de cette opposition.
La zone ident-créancier reste inchangée.

2.4.13. CDDEXI

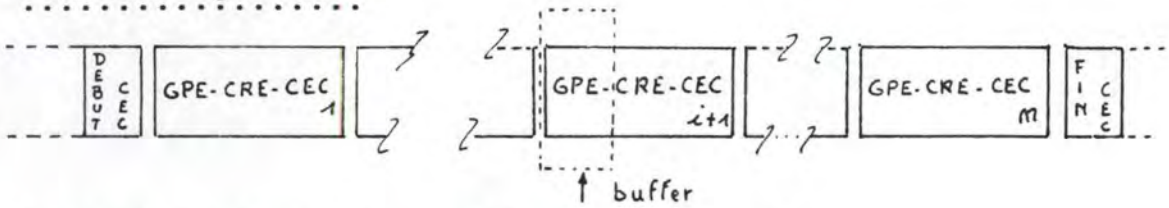
Reçoit la description d'un contrat de domiciliation (DCONTRAT) réduite au numéro de domiciliation de ce contrat (NU-DOMI) ; renvoie la description complète de ce contrat, c'est-à-dire une variable EXISTE-CDD ayant la valeur 1 si il existe un contrat de domiciliation (cdd) identifié par ce numéro de domiciliation, 0 sinon.
Dans le cas où EXISTE-CDD = 1, il renverra dans les variables NU-REF-CDD, OPPOSITION-CDD, ETAT-CDD, respectivement le numéro de référence repris dans le cdd, la valeur 1 s'il y a opposition sur le contrat (0 sinon), la valeur 1 si le cdd est révoqué (0 sinon).
Si il y a opposition sur contrat, la variable DESC-OPPOSITION-CDD contiendra la description de cette opposition. Si il y a révocation, la variable DATE-REVOCATION contiendra la date de cette révocation.
La zone NU-DOMI reste inchangée.

2.5. Construction et justification de 2 sections

2.5.1. P-GROUPE-CEC

Un groupe cec est un fichier logique cec constitué d'un enregistrement DEBUT-CEC,suivi de n ($n \geq 0$) groupes créancier-cec (CPTE-CRE-CEC), suivis d'un enregistrement FIN-CEC.

- Situation générale



Caractéristiques de la situation générale

(1) $0 \leqslant i \leqslant n$,

(2) on est au début du i+1ème groupe créancier-cec (i<n),
ou le buffer contient l'enregistrement FIN-CEC (i=n)

(3) on a effectué la génération relative aux i premiers GPE-CRE-CEC,

(4) l'entête du groupe CEC est courante.

- Initialisation

```
MOVE CORR DEBUT-CEC TO DENTETE                          I1
MOVE ORIGINE OF DEBUT-CEC TO ORIGINE OF DENTETE         I2
READ FICH-IN RECORD AT END PERFORM JAMAIS-EXECUTE       I3
```

Justification

Initialement, d'après la précondition de cette section, le buffer contient l'enregistrement DEBUT-CEC du fichier logique cec. Montrons qu'après l'exécution des instructions I1, I2, I3, les conditions (1), (2), (3) & (4) sont vérifiées (pour i = 0).

(1) OK car i = 0 et $n \geqslant 0$

(2) OK car après I3
soit le buffer contient l'enregistrement fin-cec (si n = 0)
soit on est au début du 1er groupe créancier-cec (si n > 0).

(3) OK car i = 0 et on n'a rien généré,

(4) OK car après I1 & I2,
les zones DATE-PIVOT, PROVENANCE et ORIGINE en WSS contiennent respectivement la date pivot, la provenance et l'origine de l'ensemble des descriptions d'opérations représenté par ce groupe cec.

- Arrêt

Le test de fin d'itération est ID-ENREG OF ENREG-FICH-IN = 9.
Si ce test est vérifié on a i = n et le buffer contient l'enregistrement FIN-CEC.
Il reste à exécuter

READ FICH-IN RECORD AT END MOVE 1 TO EOF-FICH-IN.

ON aura ainsi "traité" complètement le groupe-cec et on aura effectué la génération relative aux n groupes créancier-cec (constituant le groupe cec).

- Itération

On suppose la situation générale réalisée pour i ($0 \leqslant i < n$) et on cherche les instructions qui l'établissent pour i + 1

PERFORM P-GROUPE-CREANCIER-CEC I4

Justification

Après I4, selon les spécifications de la section P-GROUPE-CREANCIER-CEC, on aura traité le i+1ème GPE-CRE-CEC et on aura effectué la génération relative à ce groupe en ne modifiant pas l'entête courante.

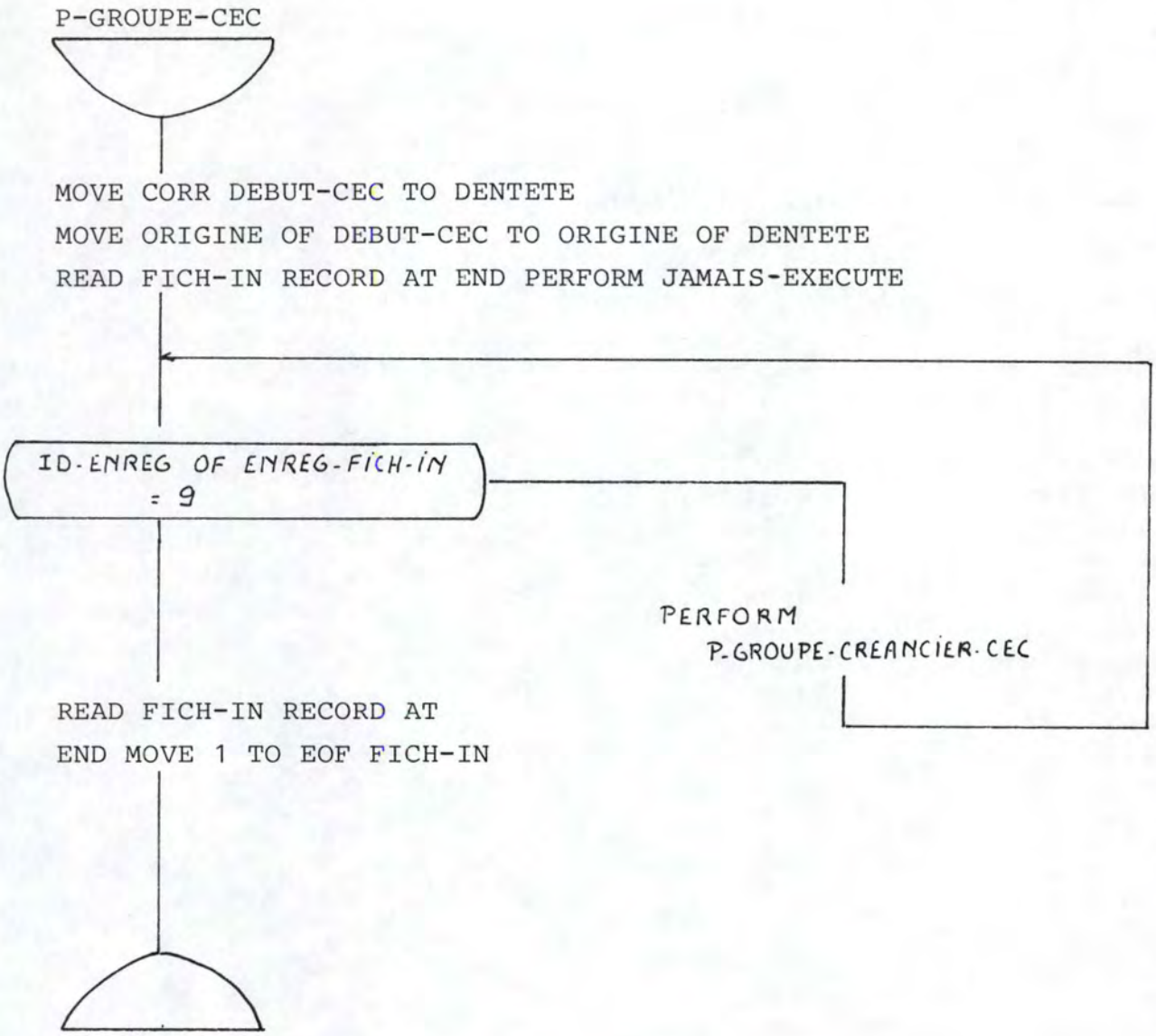
La situation générale sera vérifiée (pour i+1)

(1) OK car $0 \leqslant i < n$

(2) OK après I4 le buffer contient le Ième enregistrement suivant le dernier du groupe-cré-cec c'est-à-dire soit le premier du (i+1)+1ème GPE-CRE-CEC (i+1<n), soit l'enregistrement FIN-CEC (i+1=n).

(3) OK on a effectué la génération relative aux i premiers GPE-CRE-CEC
et l'instruction I4 effectue la génération relative au i+1ème GPE-CRE-CEC.

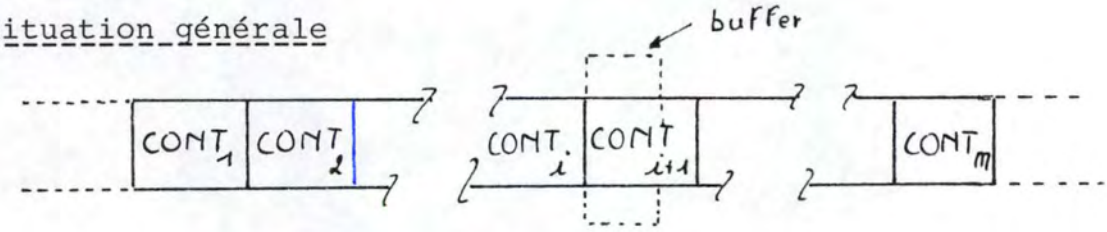
(4) OK car à l'étape i, l'entête du groupe CEC était courante et I4 ne modifie pas l'entête courante.



2.5.2. P-GROUPE-CREANCIER-CEC

Un groupe créancier-cec est une suite non vide, de taille maximale d'enregistrements OP-CEC consécutifs d'un même fichier logique CEC et ayant même numéro d'identification de créancier ; ces enregistrements sont regroupés pour former des groupes contrat-cec (CONT).

Situation générale



où CONT1,...,CONTN, sont les n groupes contrat-cec du groupe créancier-cec.

(1) $0 \leqslant i \leqslant n$,

(2) l'entête du groupe créancier-cec est courante,

(3) le nu-id-cré du groupe créancier-cec est vérifié <u>si</u>
$\exists k : 1 \leqslant k \leqslant i$ tq $CONT_k$ soit un groupe contrat-cec-dest-cger
ou non vérifié <u>sinon</u>

(4) on est au début du i+1ème groupe contrat-cec (i<n)
ou soit on est au début du groupe créancier-cec
suivant celui qu'on vient de traiter (i=n)
soit le buffer contient l'enregistrement FIN-CEC
de ce fichier logique CEC.

(5) on a effectué la génération relative aux i premiers groupes
contrat-cec de ce groupe créancier-cec.

<u>Initialisation</u>

MOVE NU-ID-CRE OF OP-CEC TO NU-ID-CRE OF DCREANCIER I1
MOVE O TO ETAT-CRE I2

<u>Justification</u>

Initialement, le buffer contient le premier enregistrement OP-CEC de ce groupe créancier-cec.
Montrons qu'après l'exécution des instructions I1, I2, les conditions (1) à (5) sont vérifiées (avec i=0).

(1) OK car i = 0

(2) OK puisqu'initialement, selon les spécifications de la section P-GROUPE-CREANCIER-CEC, l'entête du groupe créancier cec est courante.

(3) OK après l'exécution de I1, le nu-id-cré du groupe créancier-cec est courant ;
comme $\nexists k : 1 \leqslant k \leqslant 0$ tq $CONT_k$ soit un groupe contrat-cec-dest-cger, le nu-id-cré de ce groupe créancier-cec doit être non vérifié ce qui est établi après l'instruction I2.

(4) OK puisqu'initialement (selon les spécifications de la section P-GROUPE-CREANCIER-CEC) on est au début du (0+1ème) enregistrement OP-CEC de ce groupe créancier-cec

(5) OK car i=0 et on n'a rien généré.

Arrêt

Le test de fin d'itération est

ID-ENREG OF ENREG-FICH-IN = 9 OR

NU-ID-CRE OF OP-CEC NOT = NU-ID-CRE OF DCREANCIER.

Si ce test est vérifié on a i = n et le buffer contient

soit l'enregistrement FIN-CEC de ce fichier logique CEC

soit le premier enregistrement OP-CEC du groupe CREANCIER-CEC suivant celui qu'on vient de traiter.

D'après (2) et avec i = n, l'entête de ce groupe créancier cec est encore courante.

D'après (5) et avec i = n, on a effectué la génération relative aux n groupes contrat-cec de ce groupe créancier-cec.

Les spécifications de la section P-GROUPE-CREANCIER-CEC sont donc respectées.

Itération

On suppose la situation générale réalisée pour i ($0 \leqslant i < n$) et on cherche les instructions qui l'établissent pour i + 1.

```
IF DEBUT-NU-DOMI OF OP-CEC = 048                          C1
  PERFORM VERIF-P-GPE-CONT-CEC-D-CGER                     I1
ELSE PERFORM P-GROUPE-CONTRAT-CEC-DEST-CEC.               I2

VERIF-P-GPE-CONT-CEC-D-CGER.
IF CREANCIER-NON-VERIFIE                                  C2
  PERFORM VERIFICATION-NU-ID-CRE .                        I3
PERFORM P-GROUPE-CONTRAT-CEC-DEST-CGER .                  I4
```

Justification

Nous distinguons 3 types de situations générales pour les raisons suivantes :

* A l'étape i, le nu-id-cré du groupe-créancier-cec est soit vérifié, soit non vérifié et le i+ième groupe contrat-cec est soit un groupe contrat-cec-dest-cec soit un groupe contrat-cec-dest-cger.
* Lorsque le i+ième groupe contrat-cec est destiné au CEC le fait de savoir que le nu-id-cré du groupe-créancier cec est vérifié ou non n'a aucun impact sur le traitement à réaliser pour ce groupe.

Trois cas possibles peuvent donc se présenter :

1. le i+ième groupe contrat-cec est destiné au cec,
2. le nu-id-cré de ce groupe-créancier-cec est vérifié et le i+ième groupe contrat-cec est destiné à la CGER,
3. le nu-id-cré de ce groupe créancier-CEC est non vérifié et le i+ième groupe contrat-CEC est destiné à la CGER.

<u>1er cas</u> : le i+ième groupe contrat-cec est destiné au cec.

Dans ce cas, la condition C_1 est fausse et l'instruction I2 est exécutée.

Montrons qu'après son exécution, les conditions (1) et (5) sont encore vérifiées (pour i+1).

(1) OK car $0 \leqslant i < n \Rightarrow 0 \leqslant i+1 \leqslant n$

(2) OK puisque pour i, l'entête du groupe créancier-cec est courante et d'après les spécifications de la section P-GROUPE-CONTRAT-CEC-DEST-CEC, l'exécution de I2 ne modifie pas l'entête courante.

(3) OK puisque pour i, la condition (3) est vérifiée et le i+1ème groupe est un groupe contrat-cec-dest-cec.

(4) OK car après I2, selon les spécifications de la section P-GROUPE-CONTRAT-CEC-DEST-CEC, on a traité le i+ième groupe contrat-cec ; on est donc au début du (i+1)+1ème groupe contrat cec (i+1 < n),
ou soit on est au début du groupe créancier-cec suivant celui qu'on vient de traiter,
soit le buffer contient l'enregistrement FIN-CEC de ce fichier logique cec.

(5) OK car pour i on a effectué la génération relative aux i premiers groupes contrats-cec et l'exécution de I2, selon les spécifications de la section P-GROUPE-CONTRAT-CEC-DEST-CEC, effectue la génération relative au i+1ème groupe contrat-cec.

2ème cas : le nu-id-cré de ce groupe créancier-cec est véfifié et le i+1ème groupe contrat-cec est destiné à la CGER.

Dans ce cas, les conditions C1, C2 sont respectivement vraie et fausse et l'instruction I4 est exécutée.

Montrons qu'après son exécution, les conditions (1) à (5) sont encore vérifiées (pour i+1).

(1) OK car $0 \leqslant i < n \Rightarrow 0 \leqslant i+1 \leqslant n$

(2) OK puisque pour i, l'entête du groupe créancier-cec est courante, et d'après les spécifications de la section P-GROUPE-CONTRAT-CEC-DEST-CGER, l'exécution de I4 ne modifie pas l'entête courante.

(3) OK puisque pour i, le nu-id-cré de ce groupe créancier-cec est vérifié et d'après les spécifications (*) l'exécution de I4 ne modifie pas le nu-id-cré vérifié,

(4) OK car après I4, selon les spécifications(*) on a traité le i+1ème groupe contrat-cec et donc
soit on est au début du i+1ème groupe contrat-cec (i<n)
soit on est au début du groupe créancier-cec suivant celui qu'on vient de traiter, si il existe sinon le buffer contient l'enregistrement FIN-CEC de ce fichier logique CEC.

(5) OK car pour i, on a effectué la génération relative aux i premièrs groupes contrat-cec et l'exécution de I4 effectue la génération relative au i+1ème groupe contrat-cec.

3ème cas : le nu-id-cré de ce groupe créancier-cec est non vérifié et le i+1ème groupe contrat-cec est destiné à la CGER.

Dans ce cas, les conditions C_1, C_2 sont vraies et les instructions I_3, I_4 sont exécutées.

Montrons qu'après l'exécution de ces instructions, les conditions (*) (1) à (5) sont encore vérifiées (pour i+1).

(1) OK car $0 \leqslant i < n \Rightarrow 0 \leqslant i+1 \leqslant n$

(2) OK puisque pour i, l'entête du groupe créancier CEC est courante et d'après les spécifications des sections VERIFICATION-NU-ID-CRE et P-GROUPE-CONTRAT-CEC-DEST-CGER, l'exécution de I_3 & I_4 ne modifie pas l'entête courante.

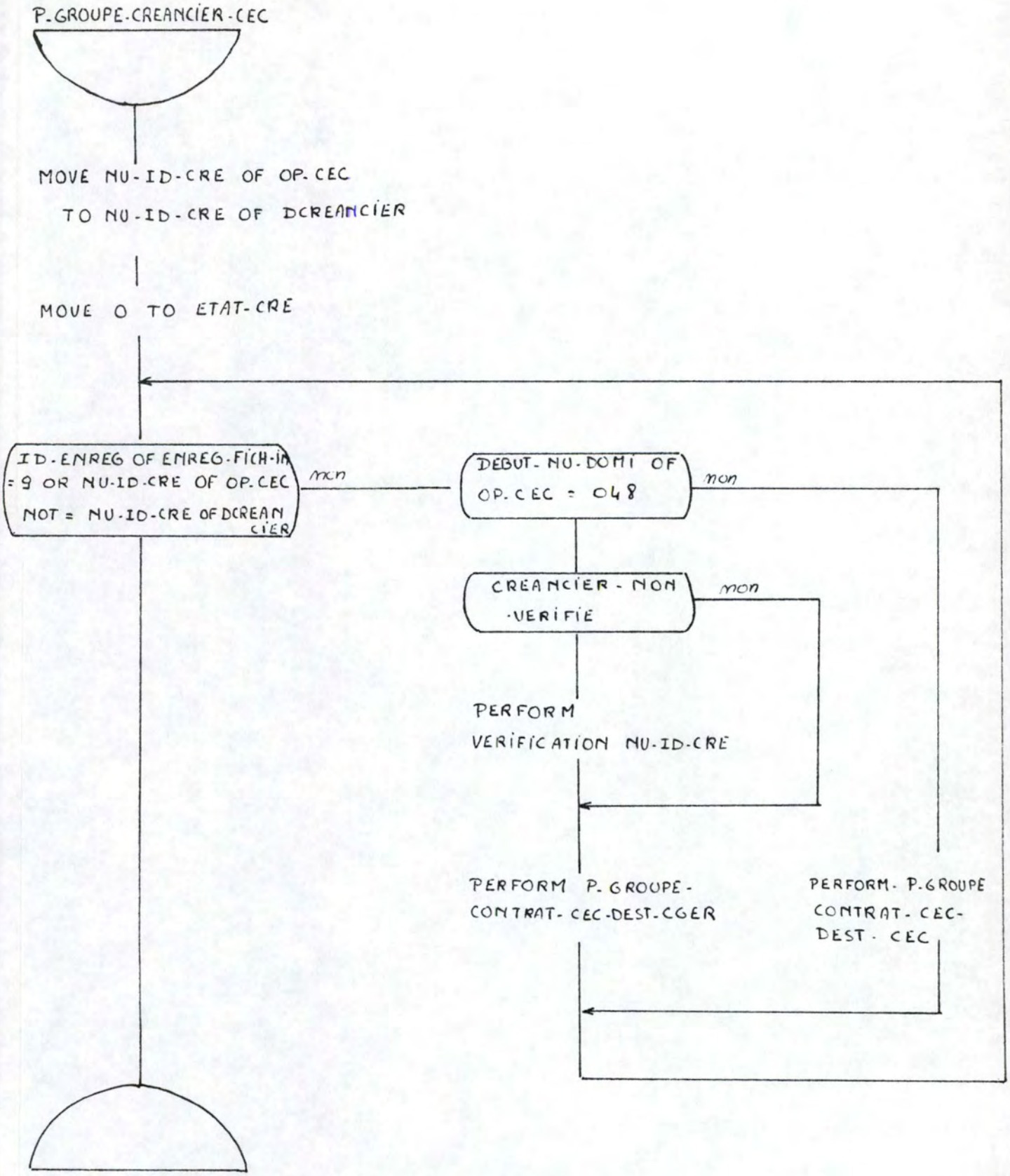
(*) de la section P-GROUPE-CONTRAT-CEC-DEST-CGER

(3) OK puisque pour i, le nu-id-cré du groupe créancier-cec est non vérifié et le i+1ième groupe contrat-cec étant destiné à la CGER, le nu-id-cré du groupe créancier-cec doit être vérifié à l'étape i+1 ; ceci est établi par l'instruction I3 et n'est pas modifié par l'exécution de I4 (selon les spécifications de la section P-GROUPE-CONTRAT-CEC-DEST-CGER).

(4) OK car selon les spécifications de la section VERIFICATION-NU-ID-CRE, l'exécution de I3 laisse inchangé l'état des fichiers et après l'exécution de I4 (selon les spécifications de la section P-GROUPE-CONTRAT-CEC-DEST-CGER) soit on est au début du i+1ème groupe contrat CEC (i<n) soit on est au début du groupe créancier cec suivant celui qu'on vient de traiter s'il existe, sinon le buffer contient l'enregistrement FIN-CEC de ce fichier logique CEC.

(5) OK car pour i on a effectué la génération relative aux i premiers groupes contrat-cec ; l'exécution de I3 laisse inchangé l'état des fichiers et celle de I4 effectue la génération relative au i+1ème groupe contrat-cec.

Le programme complet se trouve à l'annexe 5.


P.GROUPE.CREANCIER.CEC
MOVE NU-ID-CRE OF OP.CEC
TO NU-ID-CRE OF DCREANCIER
MOVE 0 TO ETAT-CRE
ID.ENREG OF ENREG.FICH.IN = 9 OR NU-ID-CRE OF OP.CEC NOT = NU-ID-CRE OF DCREANCIER
non
DEBUT.NU.DOMI OF OP.CEC = 048
non
CREANCIER.NON.VERIFIE
non
PERFORM VERIFICATION NU-ID-CRE
PERFORM P.GROUPE.CONTRAT.CEC.DEST.CGER
PERFORM.P.GROUPE CONTRAT.CEC.DEST.CEC

CHAPITRE 3 : T E S T

Nous avons appliqué ci-dessous les idées exposées au paragraphe 4 du chapitre 1 de la première partie pour essayer de construire un "bon" jeu de tests pour le programme contrôle.

1. CONSTRUCTION DU PREMIER ENSEMBLE DE JEUX DE DONNEES DE TEST

Le premier ensemble a été construit pour tester si le programme génère bien les fichiers de sortie conformes aux spécifications, quand il reçoit en entrée un fichier structuré selon la structure des entrées.
Pour ces jeux de données de test, nous supposerons par simplification que le programme réalise correctement les 7 contrôles.
Cette hypothèse (hypothèse 0) nous permettra de limiter les opérations appartenant à un groupe contrat destiné CGER à des enregistrements OP-CEC ou OP-CRE représentant des opérations acceptables.
On tentera, par la suite (but du second ensemble de jeux de données de test), de confirmer cette hypothèse.
Etant donné l'impossibilité d'effectuer des tests exhaustifs, on va essayer de dégager de l'ensemble des fichiers structurés selon la structure des entrées un échantillon qui en soit suffisamment représentatif .

1.1. Premiers jeux de données de test

Pour ce, on s'appuie sur la définition de la structure du fichier d'entrée.
Définition : "Le fichier d'entrée est une suite éventuellement vide de fichiers logiques. Un fichier logique étant soit un fichier logique CEC, soit un fichier logique créancier".

De cette définition, il résulte que le fichier d'entrée est :
soit réduit au fichier vide (premier cas),

soit réduit à un fichier logique (deuxième cas)
soit constitué de n (>1) fichiers logiques (troisième cas)

Pour tester le permier cas, nous construisons le jeu suivant :
Jeu de données de test 1
FICH-IN = au fichier vide.

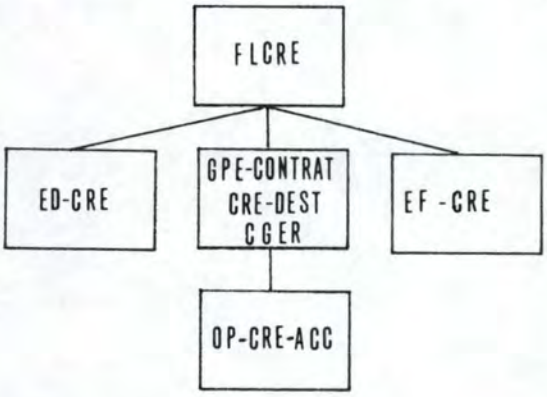
Concernant le deuxième cas,nous supposerons à ce stade que le programme est correct quand il traite un fichier d'entrée réduit à un fichier logique (hypothèse 1). Nous lèverons cette hypothèse et tenterons de la vérifier ultérieurement (cf 1.2).
Sous cette hypothèse, on va essayer de montrer que le programme est correct quand il traite un fichier d'entrée constitué de n (n>1) fichiers logiques (troisième cas).
Nous prenons arbitrairement comme valeur de n 2; car nous supposons que si le programme est correct quand il traite un fichier d'entrée constitué de n appartenant à (1,2) il le sera également pour n> 2.
Un fichier logique étant soit un fichier logique CEC, soit un fichier logique créancier et n étant égal à 2, nous aurons pour tester ce deuxième cas, 4 jeux de données de test.
Dans chacun d'eux, les fichiers logiques seront réduits à un groupe créancier pour les fichiers logiques CEC) à un seul groupe contrat CGER et à une seule opération acceptable.
Cette simplification n'est pas abusive, elle résulte des hypothèses 0 et 1.
En effet, selon l'hypothèse 0 que l'opération soit acceptable ou non n'a pas d'importance, puisqu'on suppose que le programme réalise correctement les contrôles. Selon l'hypothèse 1 que la structure du fichier logique soit la plus simple ou la plus compliquée n'a pas d'importance puisque le programme est correct quand il traite un fichier d'entrée réduit à un fichier logique.
Aussi on a choisi comme structure du fichier logique celle qui nous semblait être la plus simple sans être trop particulière (c'est-à-dire le cas où elle se réduirait au fichier vide).

Légende des schémas :

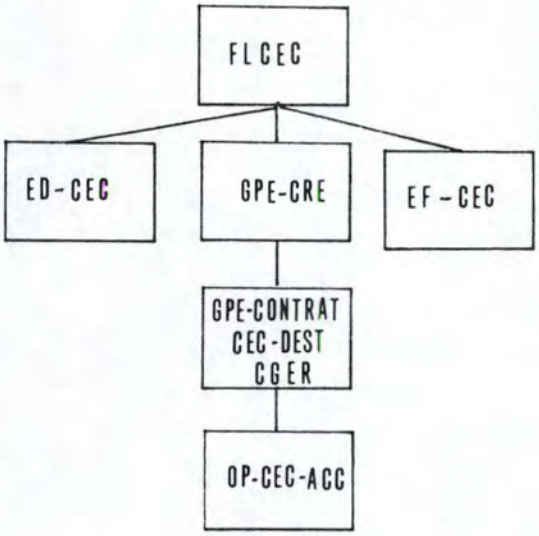
fich-in	= fichier d'entrée
flcec	= fichier logique cec
flcre	= fichier logique créancier
ed-cec	= enregistrement début-cec
ed-cre	= enregistrement debut-cre
ef-cec	= enregistrement fin-cec
ef-cre	= enregistrement fin-cre
gpe-cre-cec	= groupe créancier cec
gpe-contrat-cec-dest-cger	= groupe contrat cec destiné cger
gpe-contrat-cre-dest-cger	= groupe contrat cre destiné cger
pge-contrat-cec-dest-cec	= groupe contrat cec destiné cec
gpe-contrat-cre-dest-cec	= groupe contrat cre destiné cec
op-cec	= enregistrement op-cec
op-cre	= enregistrement op-cre
op-cec-acc	= enregistrement op-cec représentant une opération acceptable
op-cre-acc	= enregistrement op-cre représentant une opération acceptable

Dans les 4 jeux de données de test qui vont suivre,la structure d'un fichier logique sera la suivante :

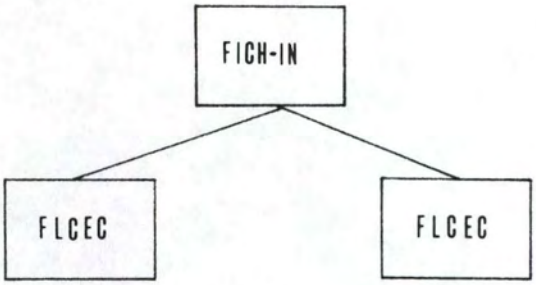
- Dans le cas d'un fichier logique créancier :



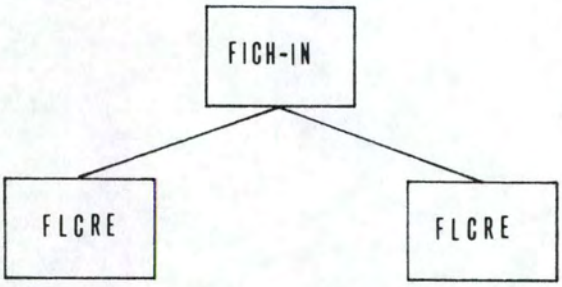
- Dans le cas d'un fichier logique cec :



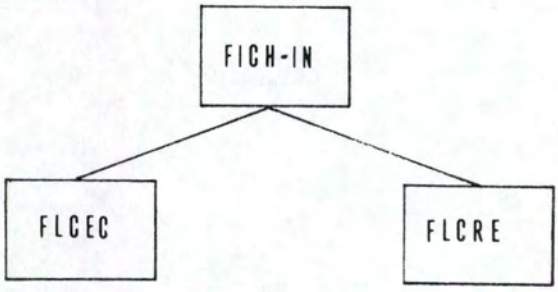
JEU DE DONNEES DE TEST 2



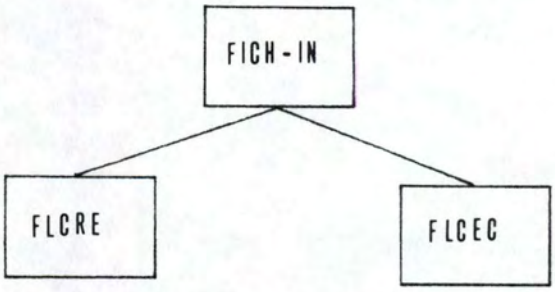
JEU DE DONNEES DE TEST 3



JEU DE DONNEES DE TEST 4



JEU DE DONNEES DE TEST 5



1.2. Contruction de jeux de données de test pour tenter de vérifier l'hypothèse 1

A présent, on va construire des jeux de données de test pour vérifier que l'hypothèse que l'on a faite précédemment est justifiée (hypothèse 1).

1.2.1. Cas où le fichier logique est un flcec

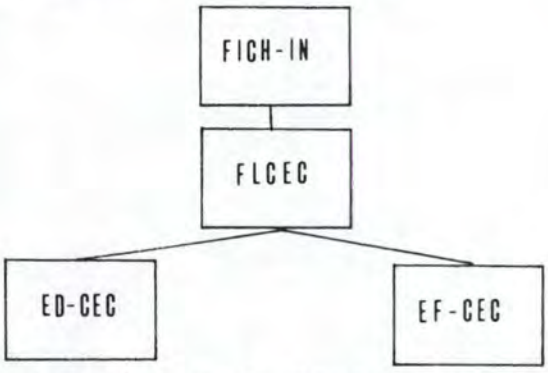
On va s'assurer que le programme est correct quand il traite un fichier d'entrée réduit à un flcec.
Pour construire les jeux de données de test, on procède comme précédemment, c'est-à-dire on part de la définition d'un fichier logique cec.

Définition : "Un fichier logique cec est constitué d'un enregistrement début-cec suivi d'une suite éventuellement vide de groupes créancier-cec suivi d'un enregistrement fin-cec".

De cette définition, il résulte qu'un fichier logique cec est

- soit réduit à un enregistrement début-cec
  suivi d'un enregistrement fin-cec (premier cas)
- soit réduit à un enregistrement début-cec
  suivi d'un seul groupe créancier-cec
  suivi d'un enregistrement fin-cec (deuxième cas)
- soit constitué d'un enregistrement début-cec
  suivi de n(>1) groupes créancier-cec
  suivi d'un enregistrement fin-cec (troisième cas)

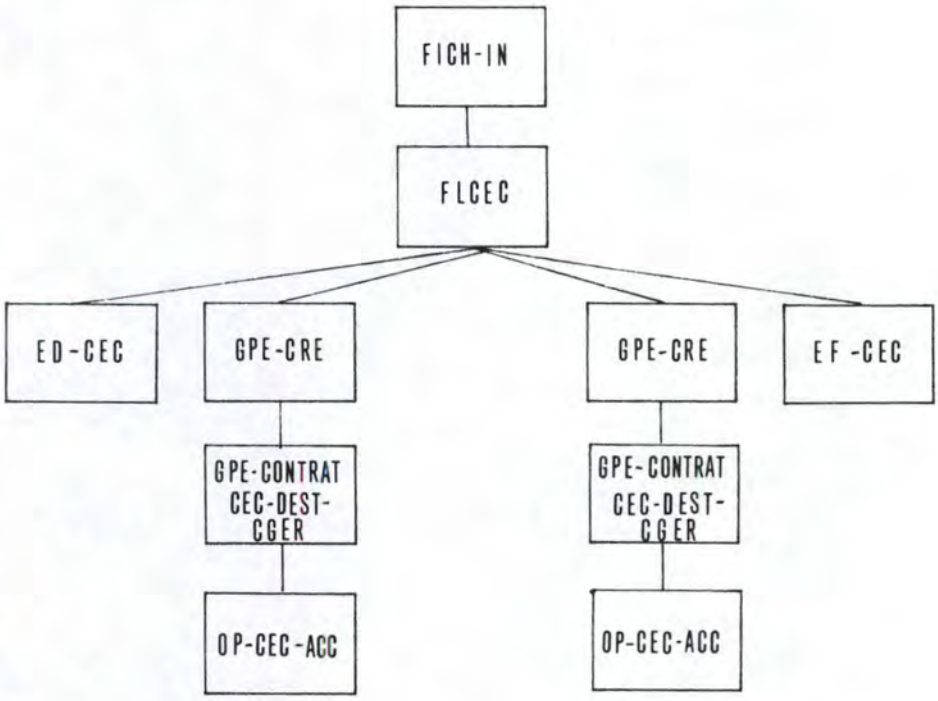
Pour tester le premier cas, nous construisons le jeu de test suivant : Jeu de données de test 6



Concernant le deuxième cas, nous supposerons par simplification que le programme est correct quand il traite un fichier d'entrée réduit à un fichier logique cec lui-même réduit à la séquence suivante : un enregistrement début-cec, un groupe créancier-cec, un enregistrement fin-cec (hypothèse 2).

Sous cette hypothèse, on va essayer de montrer que le programme est correct quand il traite un fichier constitué de n (>1) groupes créancier-cec. Pour ce faire, nous construirons un jeu de test ou le fichier d'entrée est constitué d'un fichier logique cec composé d'un enregistrement début-cec, de n (>1) groupes créancier-cec, et d'un enregistrement fin-cec.
Nous choisissons arbitrairement comme valeur de n 2 ; car nous supposons que si le programme est correct quand il traite un fichier d'entrée constitué d'un seul fichier logique cec composé d'un ed-cec, d'une suite de n appartenant à(1, 2) groupes créancier-cec, il le sera aussi pour n > 2.
Les groupes créancier-cec qui feront partie des jeux de données de test seront composés d'un seul groupe contrat-destiné-cger lui-même étant réduit à une seule opération acceptable.
Cette simplification n'est pas abusive, elle résulte des hypothèses 0 et 2.

JEU DE DONNEES DE TEST 7



Si les résultats donnés par l'exécution du programme avec ce jeu de données de test sont corrects, on pourra raisonnablement supposer que le programme est correct pour toute suite de groupe créancier-cec.

1.2.1.1. Construction de jeu de test pour tenter de vérifier l'hypothèse 2

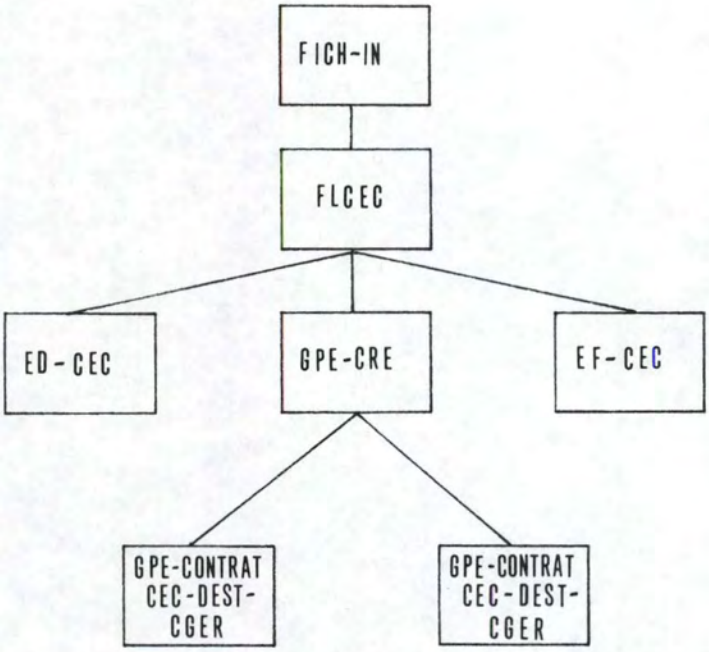
Maintenant nous allons construire des jeux de données de test pour vérifier que l'hypothèse 2 est justifiée.
C'est-à-dire que nous allons nous assurer que le programme est correct quand il traite un fichier d'entrée réduit à un fichier logique cec constitué d'un enregistrement début-cec, d'un groupe créancier-cec, et d'un enregistrement fin-cec.
Pour ce faire, nous nous basons sur la définition d'un groupe-créancier-cec.
Définition : "un groupe créancier-cec est une suite non vide de groupe contrat-cec ;
un groupe contrat-cec pouvant être destiné soit au cec soit à la cger."

De cette définition, il résulte qu'un groupe créancier-cec est
* soit constitué d'1 groupe contrat-cec (premier cas)
* soit constitué de n (>1) groupes contrat-cec (deuxième cas).

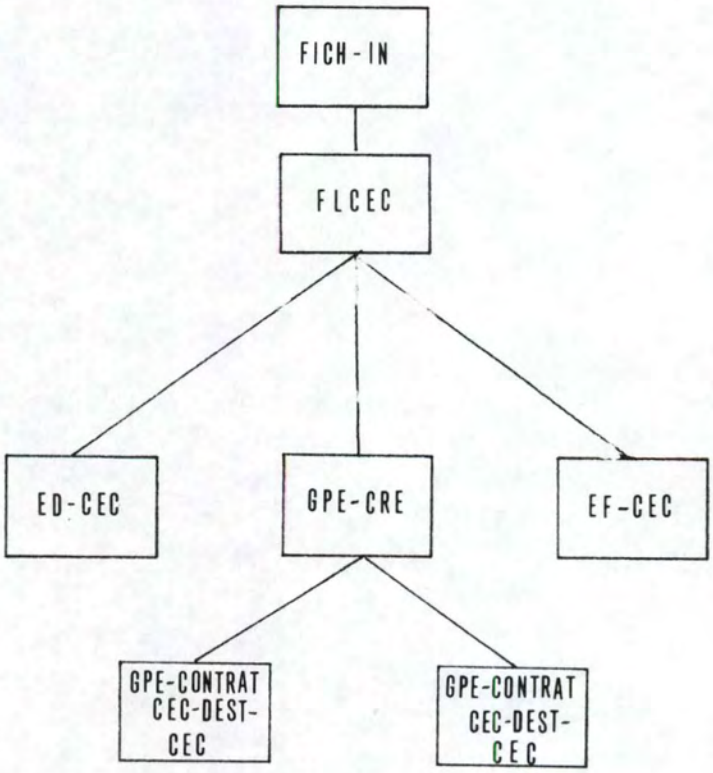
Concernant le premier cas, nous supposerons que le programme est correct quand il traite un fichier d'entrée réduit à un fichier logique cec composé d'un enregistrement début-cec, d'un seul groupe créancier-cec, d'un enregistrement fin-cec ; ce groupe créancier-cec étant réduit à un seul groupe contrat (hypothèse 3).

Sous cette hypothèse, nous allons tenter de montrer que le programme est toujours correct quand le groupe créancier-cec qu'il traite est constitué de n (>1) groupes contrat-cec. Pour ce, nous construirons des jeux de test où la valeur de n sera arbitrairement égale à 2 (la justification de ce choix est semblable à celle donnée un peu plus haut).
Un groupe contrat-cec pouvant être destiné soit au cec, soit à la cger et n étant = 2, nous aurons pour tester ce troisième cas 4 jeux de données de test.
Dans tous ces jeux de données, les groupes contrat-cec seront réduits à un seul enregistrement op-cec. Cet enregistrement op-cec, si il appartient à un groupe contrat-cec-dest-cger, représentera une opération acceptable. Cette simplificiation résulte des hypothèses 0 et 3.

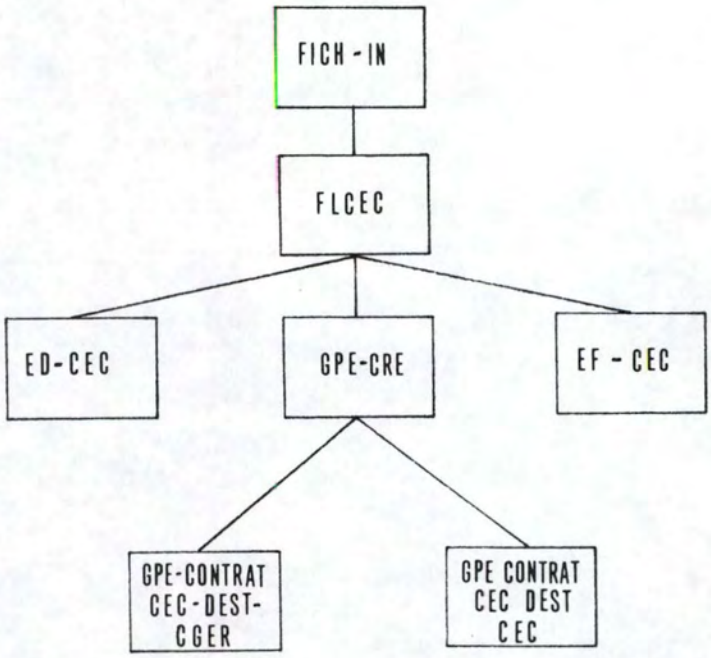
JEU DE DONNEES DE TEST 8



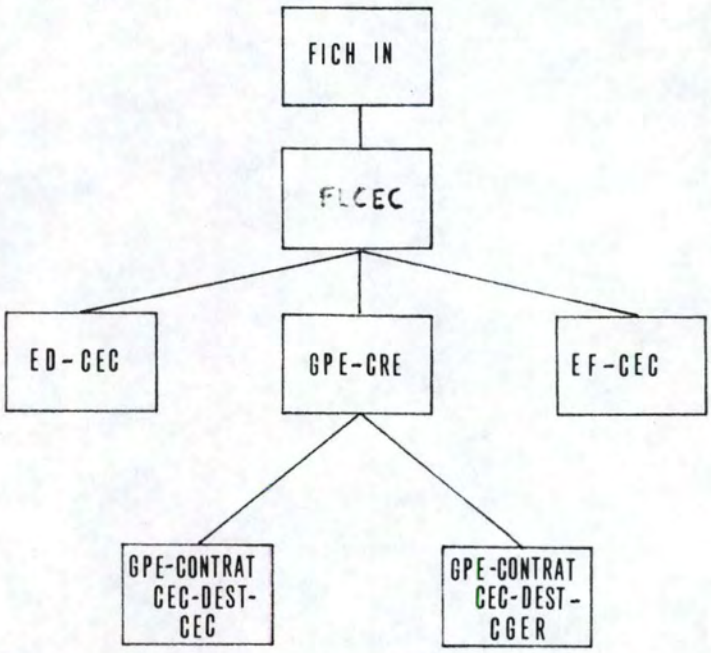
JEU DE DONNEES DE TEST 9



JEU DE DONNEES DE TEST 10



JEU DE DONNEES DE TEST 11



1.2.1.2. Construction de jeux de données de test pour tenter de vérifier l'hypothèse 3

A ce stade, nous allons construire des jeux de données de test pour vérifier que l'hypothèse 3 que l'on a faite précédemment est justifiée. Pour ce, nous nous basons sur la définition de ce qu'est un groupe contrat-cec.

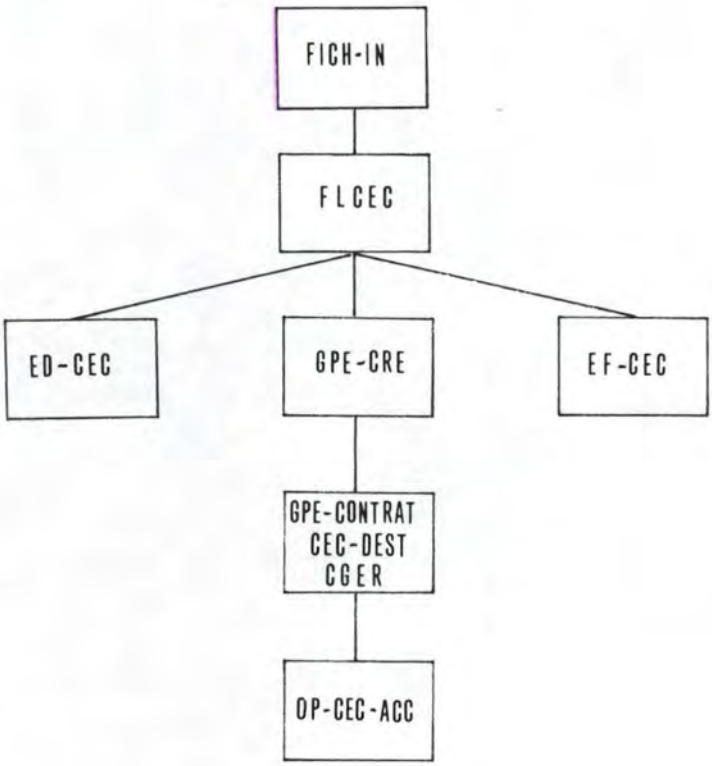
Définition : "un groupe contrat-cec, qu'il soit destiné à la cger ou au cec est une suite non vide d'enregistrements op-cec".

De cette définition, il résulte qu'un groupe contrat-cec est

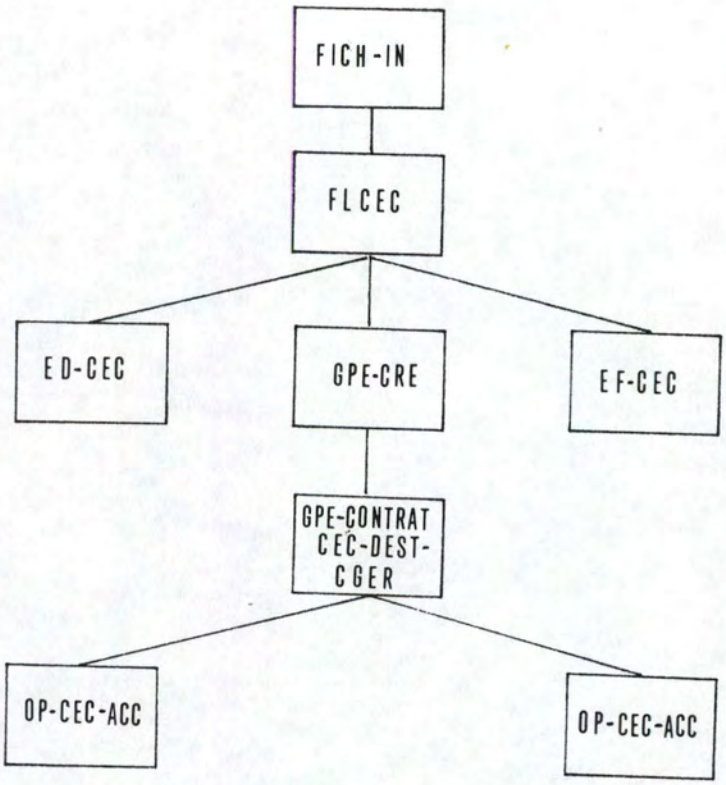
* soit un groupe contrat-cec-dest-cger et que dans ce cas il peut être - soit réduit à seul enregistrement op-cec (cas 1)
  - soit constitué de n (>1) enregistrements op-cec (cas 2)
* soit un groupe contrat-cec-dest-cec et que dans ce cas il peut être - soit réduit à un seul enregistrement op-cec (cas 3)
  - soit constitué de n (>1) enregistrements op-cec (cas 4).

Pour tester les cas 1, 2, 3, 4, nous construirons respectivement les 4 jeux de données de test suivants :

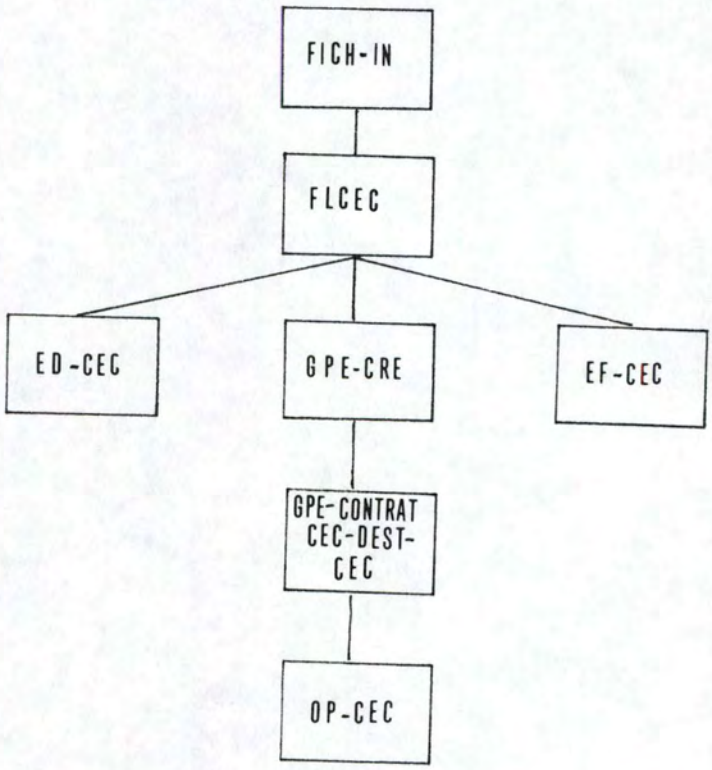
JEU DE DONNEES DE TEST 12



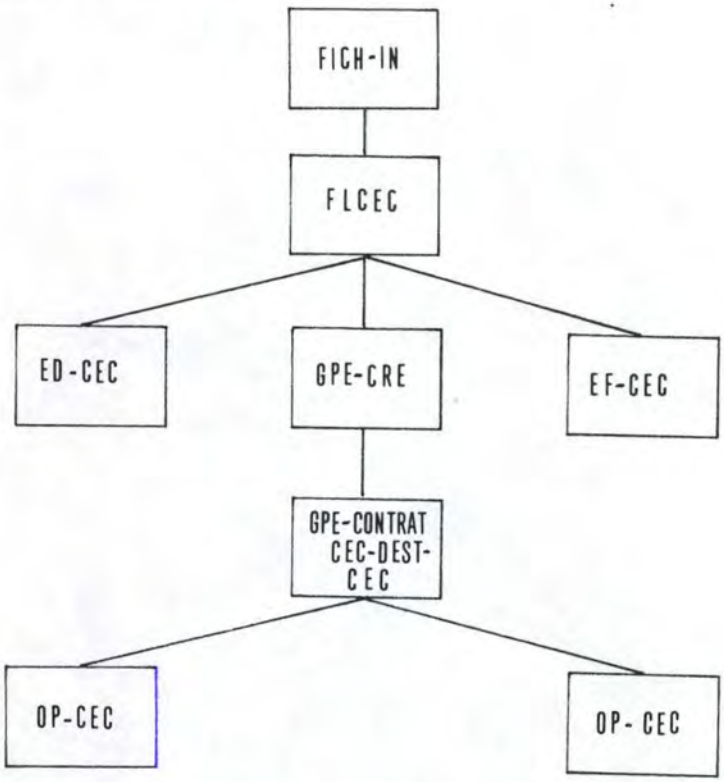
JEU DE DONNEES DE TEST 13



JEU DE DONNEES DE TEST 14



JEU DE DONNEES DE TEST 15



1.2.2. Cas où le fichier logique est un fichier logique créancier

Nous allons essayer ici de montrer que le programme est correct quand il traite un fichier d'entrée réduit à un fichier logique créancier. La structure d'un fichier logique créancier diffère de celle d'un fichier logique cec par le fait qu'il n'existe pas de notion de groupe créancier. Le raisonnement suivi pour constuire les jeux de données de test à ce niveau est semblable à celui effectué au point 1.2.1.

Sur base de la définition de ce qu'est un fichier logique créancier.

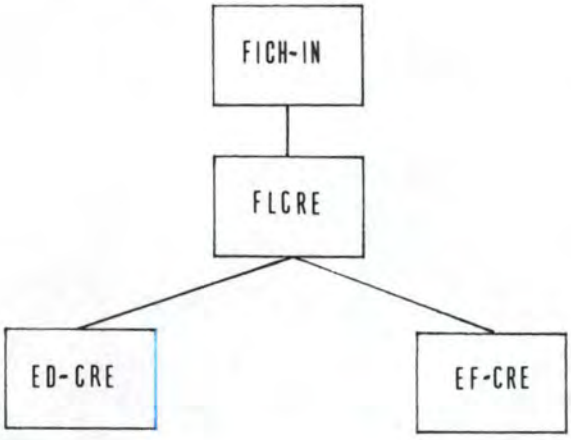
Définition : "un fichier logique créancier est constitué d'un enregistrement début-cre, suivi d'une suite éventuellement vide de groupes contrat-cre, suivi d'un enregistrement fin-cre".

On déduit qu'un fichier logique créancier peut être :

* soit réduit à un enregistrement début-cre, suivi d'un enregistrement fin-cre (cas 1)
* soit réduit à un enregistrement début-cré, suivi d'un seul groupe contrat-cre, suivi d'un enregistrement fin-cre (cas 2)
* soit constitué d'un enregistrement début-cre, suivi de n (>1) groupes contrat-cre suivi d'un enregistrement fin-cre (cas 3).

Pour tester le cas 1, nous construirons le jeu de test suivant :

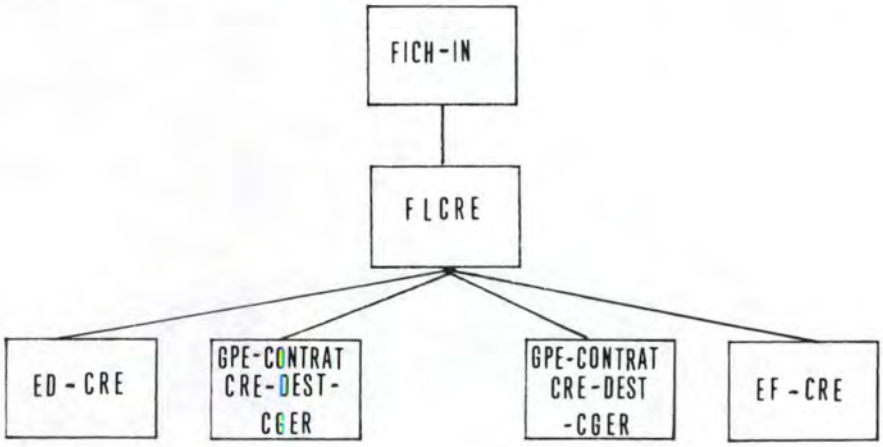
JEU DE DONNEES DE TEST 16



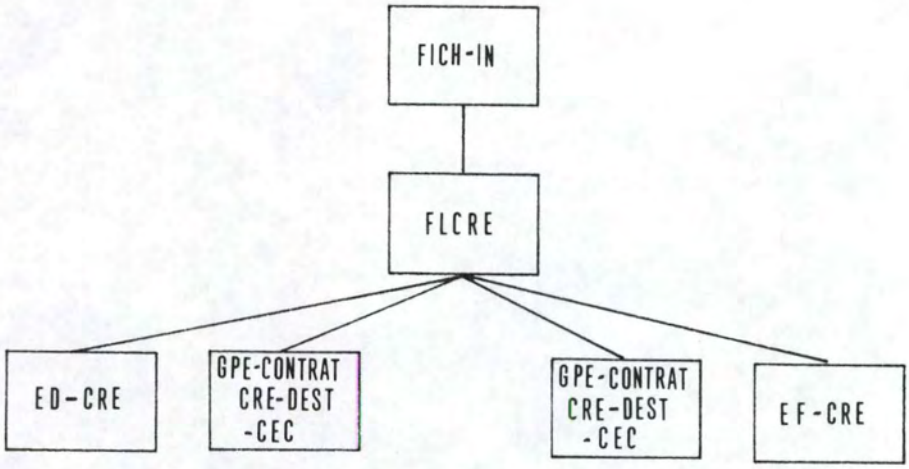
Concernant le cas 2, nous supposerons que le programme est correct quand il traite un fichier d'entrée réduit à un fichier logique créancier (hypothèse 4). Sous cette hypothèse, nous allons construire des jeux de données de test pour tenter de montrer que le programme est correct quand il traite un fichier d'entrée composé d'un fichier logique créancier constitué d'un enregistrement début-cre, suivi de n (>1) groupes contrat-cre suivi d'un enregistrement fin-cre. Nous prendrons arbitrairement comme valeur de n 2 (justification même raisonnement qu'au point 1.2.1.).

Dans tous les jeux de données de test suivant, par simplification les groupes contrat-cre seront réduits à un enregistrement op-cre (qui représentera en plus une opération acceptable pour les groupes contrat-cre-dest-cger). Cette simplification n'est pas abusive, elle est autorisée par l'existence des hypothèses 0 et 4.

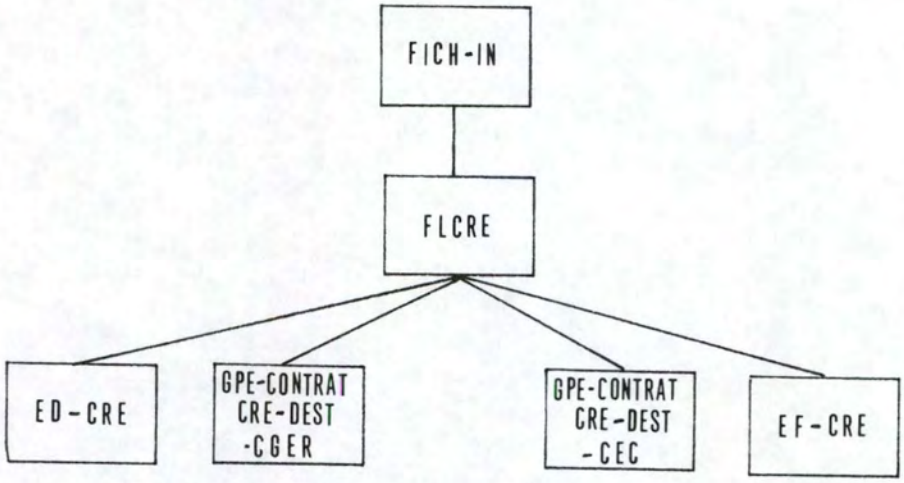
JEU DE DONNEES DE TEST 17



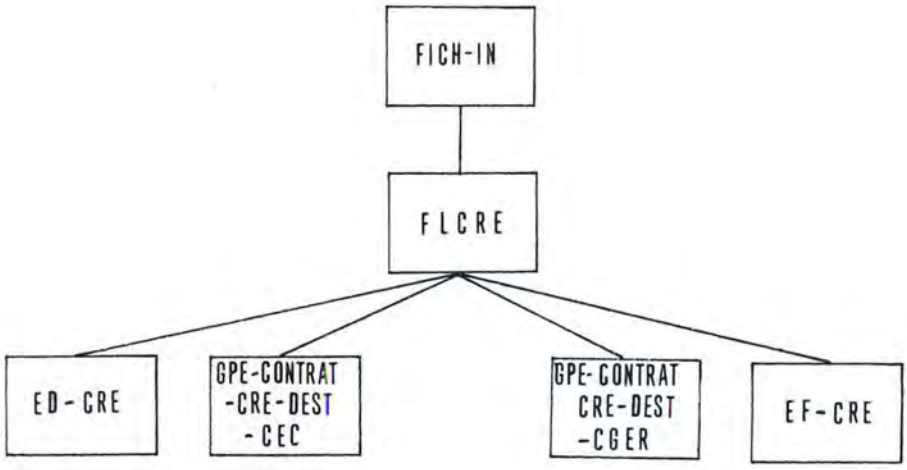
JEU DE DONNEES DE TEST 18



JEU DE DONNEES DE TEST 19



JEU DE DONNEES DE TEST 20



1.2.2.1. Construction de jeux de données de test pour tenter de montrer que l'hypothèse 4 est vérifiée

A présent, nous allons construire des jeux de données de test pour s'assurer que l'hypothèse 4 que l'on a faite précédemment est justifiée. Sur base de la définition d'un groupe contrat-cre.

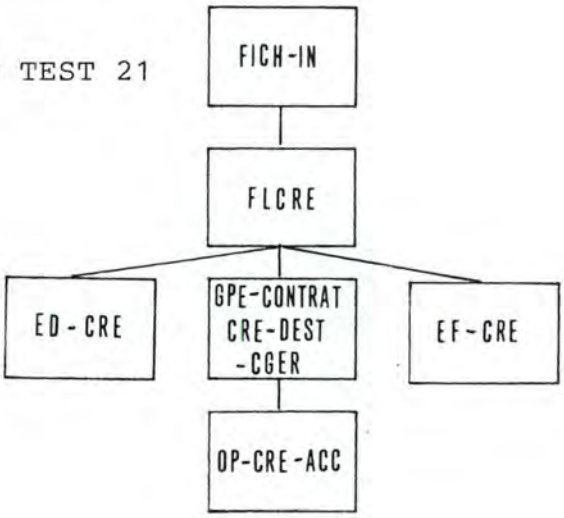
Définition : "un groupe contrat-cre qu'il soit destiné au cec ou à la cger est une suite non vide d'enregistrements op-cre".

Nous déduisons qu'un groupe contrat-cre est

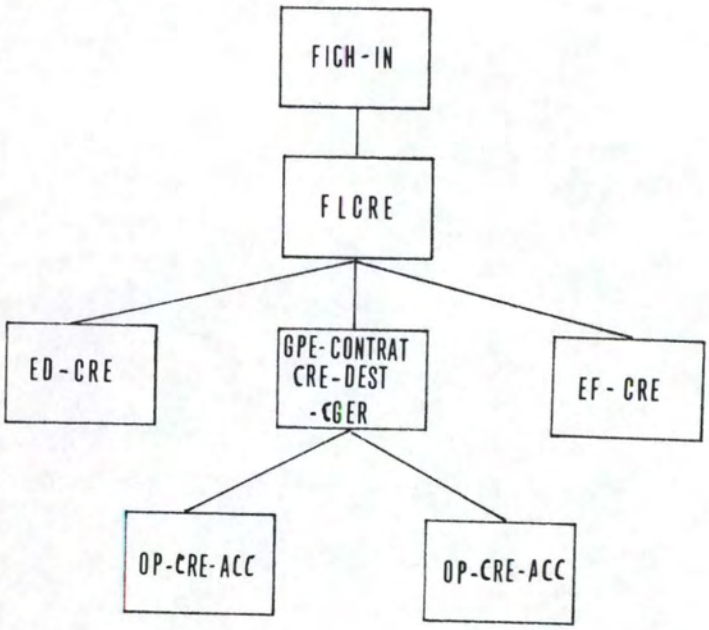
* soit destiné à la cger et que dans ce cas il est
  - soit réduit à un seul enregistrement op-cre (cas 1)
  - soit constitué de n (>1) enregistrements op-cre (cas 2)
* soit destiné à la cec et que dans ce cas il est
  - soit réduit à un seul enregistrement op-cre (cas 3)
  - soit constitué de n (>1) enregistrements op-cre (cas 4).

Pour tester les cas 1, 2, 3, 4, nous construirons respectivement les 4 jeux de test suivants en prenant arbitrairement comme valeur de n 2 :

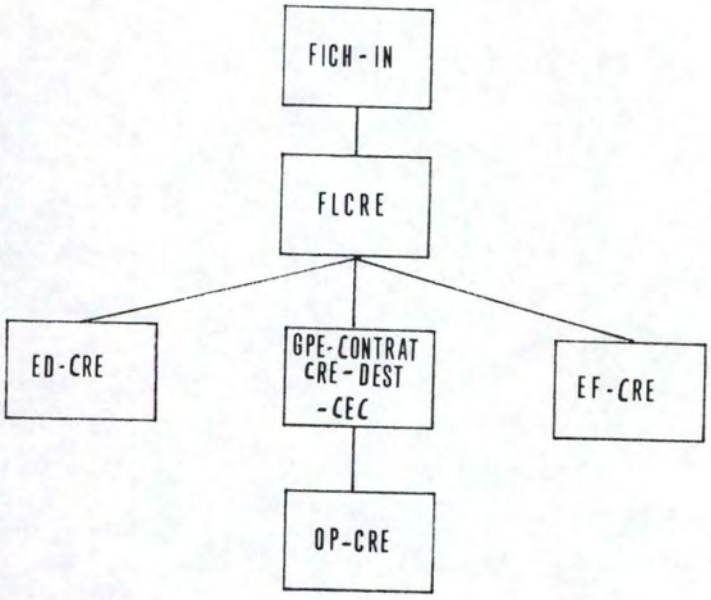
JEU DE DONNEES DE TEST 21



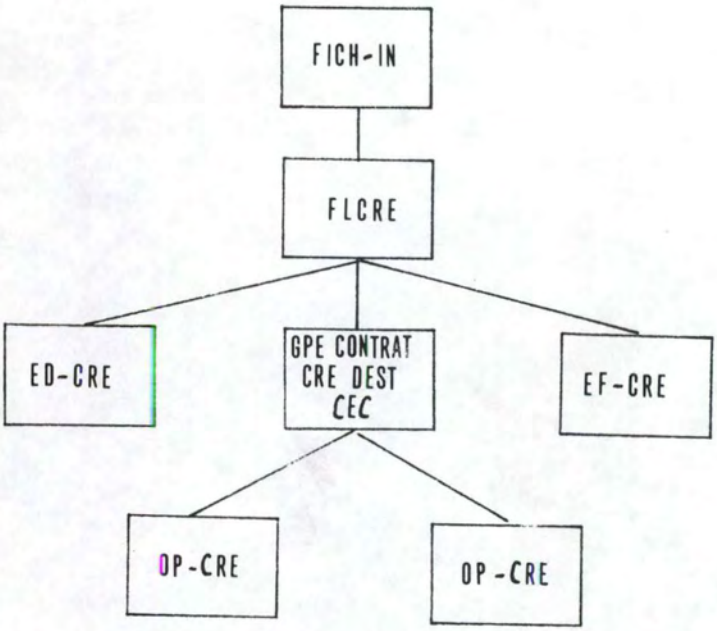
JEU DE DONNEES DE TEST 22



JEU DE DONNEES DE TEST 23



JEU DE DONNEES DE TEST 24



2. CONSTRUCTION DU SECOND ENSEMBLE DE JEU DE DONNEES DE TEST

Le second ensemble a été construit afin de tester si le programme réalise correctement les contrôles. Pour créer cet ensemble, on supposera que l'exécution du programme avec les jeux de données de test du premier ensemble n'a donné que des résultats corrects et que par conséquent on peut raisonnablement supposer que le programme est correct du point de vue des structures des fichiers d'entrée et de sortie.
La (les) structure(s) de fichier d'entrée que nous choisirons pour cet ensemble de jeux de données de test n'a donc pas beaucoup d'importance.
Pour créer cet ensemble, nous nous sommes basés sur le fait qu'il existe en réalité 2 catégories de contrôles qui peuvent être réalisés les uns indépendamment des autres.

Première catégorie

1. Numéro d'identification du créancier existe ?
2. Pas d'opposition sur ce créancier ?

Seconde catégorie

3. Numéro de domiciliation existe ?
4. Etat du contrat en cours ?
5. Pas d'opposition sur ce contrat ?
6. Compatibilité des numéros d'identification de créancier se trouvant sur le cdd et sur l'opération ?
7. Compatibilité des numéros de référence se trouvant sur le cdd et sur l'opération ?

2.1. Construction de jeux de données de test pour tenter de montrer que le programme réalise correctement les contrôles 1 et 3.

Nous supposerons d'abord que le programme réalise correctement le contrôle 2 (hypothèse 5) ainsi que les contrôles 4, 5, 6, 7 (hypothèse 6). Par la suite, nous lèverons ces hypothèses et tenterons de les vérifier (objet des sections 3.2.2. et 3.2.3.).
Nous nous arrangerons par simplicité pour avoir dans les jeux de test des opérations telles que les résultats des contrôles 2, 4, 5, 6, 7 soient positifs (dans la mesure du possible !!

En effet si le résultat du contrôle 1 est négatif, la question de savoir si le résultat du 2ème contrôle sera positif est dépourvue de sens).
Les résultats des contrôles 1 et 3 pouvant être soit positifs soit négatifs, nous construirons 4 jeux de données de test reflétant toutes les combinaisons possibles des résultats de ces deux contrôles.

JEU DE DONNEES DE TEST 25
Une description d'opération qui a satisfait aux contrôles 1 et 3.

JEU DE DONNEES DE TEST 26
Une description d'opération qui a échoué au contrôle 1 et satisfait au contrôle 3.

JEU DE DONNEES DE TEST 27
Une description d'opération qui a satisfait au contrôle 1 et échoué au contrôle 3.

JEU DE DONNEES DE TEST 28
Une description d'opération qui a échoué aux contrôles 1 et 3.

2.2. Construction de jeux de données de test pour tenter de montrer que l'hypothèse 5 est justifiée

Il s'agit ici d'essayer de montrer que le programme réalise correctement le contrôle d'existence d'une opposition sur un créancier. La question de savoir s'il y a une opposition sur un créancier n'a de sens que si le numéro d'identification de ce créancier existe.
Aussi, dans le jeu de données de test que nous construirons toutes les descriptions d'opérations devront avoir satisfaits au contrôle 1. D'autre part, nous supposerons que le programme réalise correctement les contrôles 3, 4, 5, 6, 7.
Nous pouvons donc avoir dans nos jeux de test des descriptions d'opérations qui ont satisfaits ou échoués à ces contrôles.

Par simplicité, nous choisirons des descriptions d'opérations qui ont échoué au contrôle 3, de cette manière on évite de réaliser les contrôles 4, 5, 6, 7.

JEU DE DONNEES DE TEST 29
Une description d'opération qui a satisfait aux contrôles 1 et 2 et échoué au controle 3 ; ce jeu de données de test correspond au jeu de test 27.

JEU DE DONNEES DE TEST 30
Une description d'opération qui a satisfait au contrôle 1 et échoué aux contrôles 2 et 3.

2.3. Construction de jeux de données de test pour tenter de montrer que l'hypothèse 6 est justifiée

Il faut essayer de montrer que le programme réalise correctement les contrôles 4, 5, 6, 7.
Nous supposerons qu'il en est ainsi pour le contrôle 1.
Aussi dans les jeux de données de test, que les descriptions d'opérations aient satisfait ou non au contrôle 1 ne devrait pas avoir d'importance.
Par simplicité, pour éviter de réaliser le contrôle 2, nous prendrons des descriptions d'opérations qui échouent au contrôle 1.
De plus, la question de savoir si le programme réalise les contrôles 4, 5, 6, 7 n'ayant de sens que si le numéro de domiciliation existe, les descriptions d'opération satisferont au contrôle 3.
Normalement, les résultats des contrôles 4, 5, 6, 7 pouvant être soit positifs ou négatifs, nous devrions construire 16 jeux de données de test. Nous nous contenterons de tester les cas limites et quelques cas intermédiaires.

JEU DE DONNEES DE TEST 31
Une description d'opération ayant satisfait aux contrôles 4, 5, 6, 7.

JEU DE DONNEES DE TEST 32
Une description d'opération ayant échoué aux contrôles 4, 5, 6, 7.

JEU DE DONNEES DE TEST 33
Une description d'opération ayant satisfait au contrôle 7 et échoué aux contrôles 4, 5, 6.

JEU DE DONNEES DE TEST 34
Une description d'opération ayant satisfait aux contrôles 4, 5, 6 et échoué au contrôle 7.

JEU DE DONNEES DE TEST 35
Une description d'opération ayant satisfait aux contrôles 5, 6, et échoué aux contrôles 4, 7.

JEU DE DONNEES DE TEST 36
Une description d'opération ayant satisfait aux contrôles 5, 6, 7 et échoué au contrôle 4.

CHAPITRE 4 : E R R E U R S D E T E C T E E S

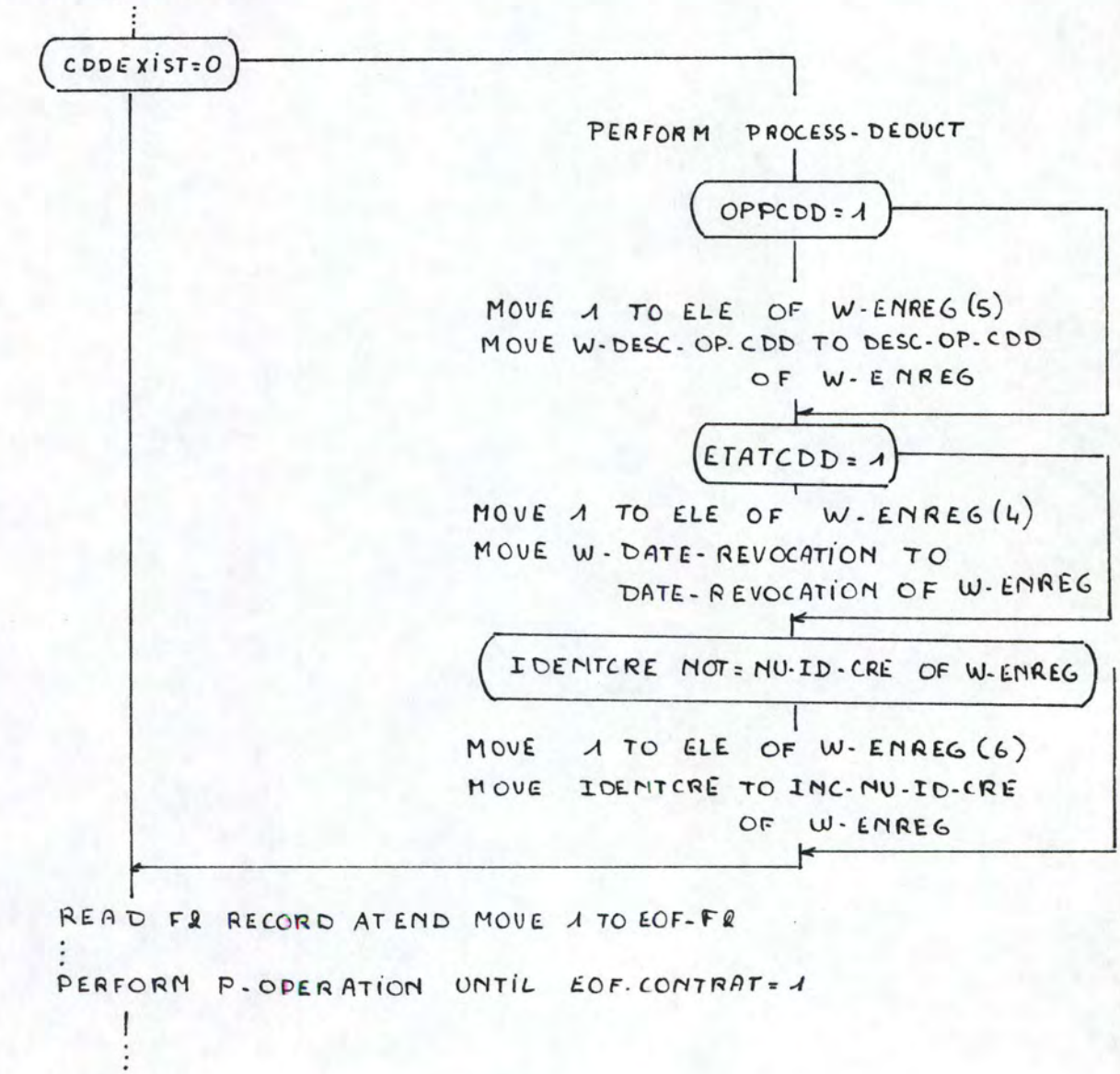
Dans ce chapitre, nous donnerons une liste des erreurs découvertes en testant les programmes avec les jeux de données de test cités au chapitre précédent. Nous montrerons qu'elles correspondent bien au type d'erreur que l'on s'attend à trouver en utilisant le principe de tests comme méthode de validation.

1. Erreurs détectées dans la première version (Jackson sans inversion)

Dans les programmes construits en suivant la méthode de Jackson (sans appliquer la technique de l'inversion) nous avons trouvé les 3 erreurs suivantes :

1.1. Erreur de codage

Cet algorithme



a été mal codé. Les 3 if successifs situés dans la branche else de la condition CDDEXIST = 0 ont été traduits par 3 if imbriqués :

```
IF CDDEXIST = 0  MOVE 1 TO ELE OF W-ENREG (3)
                 MOVE SPACE TO DEB-NU-CPTE OF W-ENREG,
                               ID-GESTION OF W-ENREG.

ELSE PERFORM PROCESS-DEDUCT

IF OPPCDD = 1  MOVE 1 TO ELE OF W-ENREG (5)
               MOVE W-DESC-OP-CDD TO DESC-OP-CDD OF W-ENREG

   ELSE NEXT SENTENCE

IF ETATCDD = 1 MOVE 1 TO ELE OF W-ENREG (4)
               MOVE W-DATE-REVOCATION TO DATE-REVOCATION OF
                                                 W-ENREG

   ELSE NEXT SENTENCE

IF IDENTCRE NOT = NU-ID-CRE OF W-ENREG

               MOVE 1 TO ELE OF W-ENREG (6)
               MOVE IDENTCRE TO INC-NU-ID-CRE OF W-ENREG.

READ F2 RECORD AT END MOVE 1 TO EOF-F2.

PERFORM P-OPERATION UNTIL EOF-CONTRAT = 1.
```

Cette erreur a été corrigée en créant un paragraphe PROCESS-CDD-EXISTANT reprenant les 3 if mentionnés ci-dessus et en les remplaçant dans la branche else de la condition CDDEXIST = 0 par un appel à ce nouveau paragraphe.

1.2. Erreur consécutive à une correction

La correction de l'erreur 1.1. a été mal réalisée. En effet, en créant le nouveau paragraphe PROCESS-CDD-EXISTANT, nous lui avons ajouté malencontreusement une instruction (PERFORM P-OPERATION UNTIL EOF-CONTRAT = 1) qui se trouvait à la suite de l'instruction IF CDDEXIST = 0..... ELSE.....

1.3. Oubli d'une instruction

Oubli d'une instruction de lecture dans un paragraphe.

2. Erreurs détectées dans la deuxième version (Jackson avec inversion)

Dans les programmes construits en suivant la méthode de Jackson en appliquant l'inversion, les erreurs suivantes ont été détectées :

2.1. Erreur de recopie

Lorsqu'on a testé PW1 et PW2 avec le jeu de données de test n° 9 (un fichier logique cec constitué de 2 groupes contrat cec destinés cec) il y a eu appel aux modules exicre, exicdd et des opérations ont été générées dans le fichier de sortie FO-ACC alors que rien de tout cela n'aurait dû avoir lieu. En fait, tout se passait comme si on traitait des groupes contrat cec destinés CGER. En examinant le code du programme, on a pu constater qu'effectivement un groupe contrat cec destiné cec était traité comme s'il s'agissait d'un groupe contrat cec destiné CGER.

Partie du programme erronée :

P-GROUPE-CONTRAT-CEC.

IF DEBUT-NU-DOMI OF OP-CEC NOT = 048 GO TO P-GPE-CONTRAT-CEC-
DEST-~~CGER~~
CEC

Cette erreur est due au fait que certains paragraphes se ressemblant assez fort, après en avoir créé un, on l'a recopié en lui apportant par la suite quelques modifications. Malheureusement toutes les modifications telle celle citée ci-dessus n'ont pas été faites.

3. Erreurs détectées dans la troisième version (explicitation des raisonnements)

Dans les programmes construits en suivant la méthode

"explicitation des raisonnements" les principales erreurs sont survenues parce que nous n'avions pas pris la peine de spécifier les sections calcul-desc-complétée-cec, calcul-desc-complétée-cré.

4. Commentaires

4.1. Peu d'erreurs : le fait d'expliciter les raisonnements fait décroitre très fortement le nombre d'erreurs et réduit considérablement la "mise au point".

4.2. La plupart des erreurs rencontrées ne sont pas des erreurs de conception mais des fautes "accidentelles" : fautes de recopie, oublis d'instructions. C'est bien ce genre d'erreurs qu'on s'attendait à rencontrer (voir paragraphe 4 du chapitre 1 de la 1ère partie).

4.3. Les erreurs plus fondamentales rencontrées dans le cas du programme construit par la méthode basée sur l'"explicitation des raisonnements" se sont présentées lorsqu'on a omis d'appliquer les principes proposés en n'accordant pas assez d'importance à la spécification de certaines parties.

T R O I S I E M E P A R T I E

TENTATIVE D'EVALUATION DE LA DEMARCHE PROPOSEE

ET COMPARAISON AVEC LES DEMARCHES CLASSIQUES

1. SPECIFICATIONS

En élaborant la deuxième partie de ce travail et plus particulièrement le chapitre 1, nous avons rencontré d'énormes difficultés concernant la compréhension du problème à traiter. Les dossiers que nous avons reçus étaient pour la plupart constitués de diagrammes dépourvus de toute explication, des questions centrales comme celle des contrôles à réaliser n'étaient pas abordées en détail, certaines incohérences apparaissaient entre dossiers et aucun document ne permettait de les comprendre. En fait, certaines justifications ont pu être obtenues auprès des personnes responsables de l'application. (Il est à remarquer que la personne la plus compétente en la matière était temporairement engagée à la CGER et qu'actuellement elle n'y travaille plus. Dès lors, où pourra-t-on trouver les justifications dont on aurait éventuellement besoin ?, pas dans les dossiers en tous les cas).
On pourrait nous rétorquer que notre difficulté provenait d'une mauvaise connaissance du milieu bancaire et de l'environnement dans lequel cette application devait s'insérer.
Nous pensons, cependant, que cette connaissance n'aurait pas suffit : les descriptions techniques, quasi incompréhensibles le seraient restées et l'évaluation de l'importance de certaines remarques n'aurait pas été plus facile. C'est pourquoi nous croyons que la présentation raisonnée est plus satisfaisante : elle oblige à expliciter les raisonnements et permet d'avoir une compréhension plus complète du problème.
Cependant son utilisation n'est pas toujours aisée, elle exige que l'on soit à même de s'exprimer avec clarté et concision. C'est ce que nous avons essayé de faire dans la 2ème partie de ce travail. Ce que nous proposons n'est toutefois, peut-être pas parfait et il est sans doute possible de faire mieux.
Ajoutons que cette méthode n'est pas incompatible avec les idées de l'analyse fonctionnelle. Elle pourrait être utilisée, par exemple, pour justifier le schéma conceptuel. Dans ce cas, on veillera à donner une signification précise à chacun des éléments du schéma conceptuel et à justifier en quoi ils sont nécessaires (et suffisants) à la réalisation de l'application.

2. CONSTRUCTION DES PROGRAMMES

La méthode de Jackson n'est pas complètement systématique. La détermination des correspondances ne suffit pas pour construire le programme. Les conditions et les actions primitives doivent être précisées. Cette dernière tâche n'est pas toujours facile à réaliser dans la mesure où les composants (de l'arbre représentant la structure du programme) n'ont pas été spécifiés ou l'ont été de manière imprécise. D'où l'intérêt de combiner la méthode de Jackson avec celle mettant l'accent sur les spécifications. Dans ce cas, la principale utilité de la méthode de Jackson est qu'elle permet d'aider à trouver une bonne découpe du problème en sous-problèmes. Cependant, la méthode de Jackson ne constitue réellement une aide que pour certains types de problèmes. Pour d'autres, elle peut être particulièrement artificielle. A ce sujet, nous pouvons donner comme exemple le programme P1 (paragraphe 1.5. du chapitre 2 de la 2ème partie).
Nous avons cherché à mettre sur le flux intermédiaire, une structure qui corresponde avec celle des entrées ; pour ce, nous avons travaillé quelque peu à l'envers puisque nous nous sommes basés sur une idée intuitive de ce que pourrait bien être la structure du programme.
En fait, ce problème se présente souvent lorsque les structures des entrées et des sorties du problème à traiter sont "très incompatibles".

3. TESTS

Les tests construits au chapitre 3 de la 2ème partie, correspondent à des tests black box à ceci près qu'ils sont justifiés. La "théorie" des tests black box ne dit pas comment choisir les bons jeux de test, parce que cela dépend du problème. Nous avons essayé de justifier le choix de ces jeux de test en fonction du problème en formulant des hypothèses sous lesquelles ils sont équivalents à des tests exhaustifs. La difficulté inhérente à cette façon de faire est double :

- les hypothèses en question sont parfois vagues.
  Exemple : un programme traite tous les enregistrements à partir du 2ème de la même façon,

- elles ne sont pas toujours réalistes.
  Exemple : il n'y a pas une totale indépendance entre la construction du premier et second ensembles de jeux de données de tests : la bonne réalisation des contrôles a une influence sur la génération des sorties.

REFERENCES

(BOD.PIG., 83) F. BODART et Y. PIGNEUR : Conception assistée des applications informatiques
1. Etude d'opportunité et analyse conceptuelle
Masson, Paris (1983)

(CLARINVAL, 81) A. CLARINVAL : Comprendre, connaître et maîtriser le cobol, norme ansi cobol 1974. Presses universitaires de Namur 1981.

(DEVILLE, 83) Y. DEVILLE : Spécification, construction et validation d'un analyseur fortran interactif. Mémoire présenté en juin 83.

(DIJKSTRA, 72) Edsger W. Dijkstra : "Why correctness must be a mathematical concern", 1972.

(DIJKSTRA, 76) Edsger W. Dijkstra : "A discipline of Programming" Prentice Hall, Englewood Cliffs (N.Y.), 1976.

(GRAAS, 85) C. Graas : Jackson Structured Programming.

(HAINAUT, 85) J.L. Hainaut : Conception assistée des applications informatiques.
2. Conception de la base de données
Masson, Paris (1985).

(JACKSON, 75) M.A. Jackson : Principles of Program Design Academic Press Inc. (London) LTD, 1975

(LE CHARLIER, 80) B. Le Charlier : Définition du langage LSD80 et spécifications des procédures de l'interpréteur du langage LSD80. Non publié, 1980.

(LE CHARLIER, 83) B. Le Charlier : L'ordinateur est-il infaillible ? La revue nouvelle.

(LE CHARLIER, 83) B. Le Charlier : Séminaire de programmation Notes de cours, 1983.

(LE CHARLIER, 85) B. Le Charlier : Réflexions sur le problème de la correction des programmes Thèse de doctorat, FNDP Namur, 1985.

(LEROY, 75) H. LEROY : La fiabilité des programmes. Presses universitaires de Namur, 1975.

(LEROY, 78) H. LEROY : La fiabilité des programmes. Notes de cours, Ecole d'été de L'AFCET, 1978.

(LEROY, 80) H. LEROY : Qu'est qu'un programme correct ? Non publié, 1980.

(LEROY, 85) H. LEROY : Théorie de la calculabilité. Notes de cours, 1985.

(MEYER, BAUDOIN, 84) B. Meyer et C. Baudoin : Méthodes de programmation. Editions Eyrolles, 1984.

(PARNAS, 72a) D.L. Parnas : A Technique for Software Module Specification with Examples. Comm. ACM 15 (5), May 1972, 330-336.

(PARNAS, 72b) D.L. Parnas : On the Criteria Be Used in Decomposing Systems into Modules. Comm.ACM 15(12), December 1972, 1053-1058

(VAN LAMSWEERDE, 82) A. Van Lamsweerde : Automatisation de la production de logiciels d'applications : quelques approches. 1ère partie . T.S.I. Technique et Science informatiques 1 (6) Nov-Déc 1982, 475-494.

(VAN LAMSWEERDE, 83a) A. Van Lamsweerde : Automatisation de la production de logiciels d'application : quelques approches. 2ème partie T.S.I. Technique et Science informatique 2 (1), Janv. Fév. 1983, 5-20.

(VAN LAMSWEERDE, 83b) A. Van Lamsweerde : Programmation d'applications de gestion et langage cobol. Notes de cours 1983.

(VAN LAMSWEERDE, 84) A. Van Lamsweerde : Méthodologie de développement de logiciels. Notes de cours 1984.

* * *

A N N E X E 1

DOM 80	ANNEXE 1	MOD. 611

AVIS DE DOMICILIATION

Les 3 exemplaires de cet avis doivent être remis à l'institution financière | A

Le soussigné
(au nom de qui les factures sont établies)

nom : .....
rue, n° : ..... boîte : ...
code postal, localité : 1

prie la firme
(qui établit les factures)

nom : .....
rue, n° : ..... boîte ...
code postal, localité : 2

d'encaisser à partir de ce jour et jusqu'à révocation expresse toutes factures portant

référence ou n° d'abonné | | | | | | | | | | | | | | | | | | | | 3

auprès de l'institution financière

nom : .....
rue, n° : ..... boîte ....
code postal, localité : 4

par le débit du compte n° | | | | - | | | | | | | - | | | 5

au nom _ du soussigné

_ de (*)

nom : .....
rue, n° : ..... boîte ....
code postal, localité : 6

Localité : ..... Date : .....
Pour accord, le titulaire de compte (*). Signature,

(*) Exige seulement si le titulaire de compte n'est pas le destinataire des factures

RESERVE A L'INSTITUTION DOMICILE

La domiciliation mentionnée ci-dessus a été acceptée sous le numéro

NUMERO DE DOMICILIATION

| | | | - | | | | | | | | - | | | 7

Numéro d'identification du créancier

| | | | | | | | | | | | | 8

Date : 9
Cachet et signature

A N N E X E 2

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. P1.
*

*
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT FICH-IN ASSIGN TO DSK
      RECORDING MODE IS ASCII
      ORGANIZATION IS SEQUENTIAL.
     SELECT F2      ASSIGN TO DSK
      RECORDING MODE IS ASCII
      ORGANIZATION IS SEQUENTIAL.
*
*
 DATA DIVISION.
*
 FILE SECTION.
 FD  FICH-IN; LABEL RECORDS ARE STANDARD
VALUE OF ID IS  NON-EXT-FICH-IN.
 01  DEBUT-CEC.
     02  ID-ENREG      PIC 9.
     02  DATE-PIVOT    PIC 9(6).
     02  PROVENANCE    PIC 9(3).
 01  DEBUT-CRE.
     02  ID-ENREG      PIC 9.
     02  FILLER        PIC X(12).
     02  DATE-PIVOT    PIC 9(6).
     02  FILLER        PIC X(15).
     02  PROVENANCE    PIC 9(11).
     02  NU-ID-CRE     PIC 9(11).
     02  NU-CPTE-CRE   PIC 9(12).
 01  OP-CEC.
     02  ID-ENREG      PIC 9.
     02  FILLER        PIC X(21).
     02  NU-DOMI.
        03  DEBUT-NU-DOMI     PIC 9(3).
        03  FIN-NU-DOMI       PIC 9(9).
     02  NU-CPTE-CRE   PIC 9(12).
     02  NATURE-OP     PIC 9.
     02  MONTANT-OP    PIC 9(12).
     02  NOM-CRE       PIC X(26).
     02  NU-ID-CRE     PIC 9(11).
     02  COMM1 PIC X(15).
     02  COMM2 PIC X(15).
     02  NU-REFERENCE PIC 9(12).
 01  OP-CRE.
     02  ID-ENREG      PIC 9.
     02  FILLER        PIC X(6).
     02  NU-DOMI.
         03  DEBUT-NU-DOMI    PIC 9(3).
         03  FIN-NU-DOMI      PIC 9(9).
     02  NATURE-OP     PIC 9.
     02  MONTANT-OP    PIC 9(12).
     02  NOM-CRE       PIC X(26).
     02  COMM1 PIC X(15).
     02  COMM2 PIC X(15).
     02  NU-REFERENCE PIC 9(12).
 01  ENREG-FICH-IN.
```

```
     02 TD-ENREG        PIC 9.
*
 FD F2; LABEL RECORDS ARE STANDARD
 VALUE OF ID IS NOM-EXT-F2.
 01  ENTETE-CREANCIER.
     02  ID-ENREG       PIC 9.
     02  DATE-PIVOT     PIC 9(6).
     02  PROVENANCE     PIC 9(11).
     02  NU-ID-CRE      PIC 9(11).
 01  ENTETE-CONTRAT.
     02  ID-ENREG       PIC 9.
     02  NU-DUMI        PIC 9(12).
 01  OPERATION.
     02  ID-ENREG       PIC 9.
     02  NATURE-OP      PIC 9.
     02  MONTANT-OP     PIC 9(12).
     02  NOM-CRE        PIC X(26).
     02  COMM1 PIC X(15).
     02  COMM2 PIC X(15).
     02  NU-REFERENCE PIC 9(12).
     02  NU-CPTE-CRE  PIC 9(12).
 01  ENREG-F2.
     02  ID-ENREG       PIC 9.
*
*
 WORKING-STORAGE SECTION.
*
 01  W-ENTETE-CREANCIER.
     02  ID-ENREG       PIC 9 VALUE IS 0.
     02  DATE-PIVOT     PIC 9(6).
     02  PROVENANCE     PIC 9(11).
     02  NU-ID-CRE      PIC 9(11).
 01  W-ENTETE-CONTRAT.
     02  ID-ENREG       PIC 9  VALUE IS 2.
     02  NU-DUMI        PIC 9(12).
 01  W-OPERATION.
     02  ID-ENREG       PIC 9  VALUE IS 1.
     02  NATURE-OP      PIC 9.
     02  MONTANT-OP     PIC 9(12).
     02  NOM-CRE        PIC X(26).
     02  COMM1 PIC X(15).
     02  COMM2 PIC X(15).
     02  NU-REFERENCE PIC 9(12).
     02  NU-CPTE-CRE  PIC 9(12).
 01  EOF-CREANCIER-CEC PIC 9.
 01  EOF-CONTRAT       PIC 9.
 01  EOF-FICH-IN       PIC 9.
 01  ENTETE-IMPRIMEE   PIC 9.
 77  NOM-EXT-FICH-IN   PIC X(9).
 77  NOM-EXT-F2 PIC X(9).
*
*
*
 PROCEDURE DIVISION.
*
*
 PROCESS-FICH-IN.
     PERFORM LIRE-NOM-FICH.
     OPEN INPUT FICH-IN.
     OPEN OUTPUT F2.
```

```
     [illegible] EOF-FICH-IN.
     READ FICH-IN RECORD AT END MOVE 1 TO EOF-FICH-IN.
     PERFORM PGROUPE UNTIL EOF-FICH-IN = 1.
     CLOSE FICH-IN,F2.
     STOP RUN.
*
 LIRE-NOM-FICH.
     DISPLAY  "VEUILLEZ DONNER LES NOMS EXTERNES DES FICHIERS".
     DISPLAY  "MAX 9 CHAR : FFFFFFSSS DEVIENDRA FFFFFF.SSS".
     DISPLAY  "FICHIER FICH-IN : " WITH NO ADVANCING.
     ACCEPT  NOM-EXT-FICH-IN.
     DISPLAY  "FICHIER F2 : " WITH NO ADVANCING.
     ACCEPT  NOM-EXT-F2.
*
 PGROUPE.
     IF PROVENANCE OF DEBUT-CEC = 102  PERFORM  P-GROUPE-CEC
 ELSE  PERFORM  P-GROUPE-CREANCIER.
*
 P-GROUPE-CEC.
     MOVE CORR DEBUT-CEC TO W-ENTETE-CREANCIER.
     READ FICH-IN RECORD AT END PERFORM JAMAIS-EXECUTE.
     PERFORM  P-GROUPE-CREANCIER-CEC
     UNTIL ID-ENREG OF ENREG-FICH-IN  =  9.
     READ FICH-IN RECORD AT END MOVE 1 TO EOF-FICH-IN.
*
 P-GROUPE-CREANCIER-CEC.
     MOVE 0 TO EOF-CREANCIER-CEC.
     MOVE  NU-ID-CRE OF OP-CEC  TO  NU-ID-CRE OF W-ENTETE-CREANCIER.
     MOVE 0 TO ENTETE-IMPRIMEE.
    PERFORM P-GROUPE-CONTRAT-CEC UNTIL  EOF-CREANCIER-CEC = 1.
*
 P-GROUPE-CONTRAT-CEC.
     IF DEBUT-NU-DOMI OF OP-CEC  = 048
   PERFORM P-GPE-CONTRAT-CEC-DEST-CGER
     ELSE PERFORM P-GPE-CONTRAT-CEC-DEST-CEC.
*
 P-GPE-CONTRAT-CEC-DEST-CGER.
     IF ENTETE-IMPRIMEE = 0  WRITE  ENTETE-CREANCIER FROM W-ENTETE-CREANCIER
                             MOVE 1 TO ENTETE-IMPRIMEE.
     MOVE 0 TO EOF-CONTRAT.
     MOVE NU-DOMI OF OP-CEC  TO  NU-DOMI OF W-ENTETE-CONTRAT.
     WRITE ENTETE-CONTRAT FROM W-ENTETE-CONTRAT.
     PERFORM P-OPERATION-CEC-DEST-CGER UNTIL EOF-CONTRAT = 1.
*
 P-OPERATION-CEC-DEST-CGER.
     MOVE CORR OP-CEC TO W-OPERATION.
     WRITE OPERATION FROM W-OPERATION.
     READ FICH-IN RECORD AT END PERFORM JAMAIS-EXECUTE.
     IF ID-ENREG OF ENREG-FICH-IN = 9
        MOVE 1 TO EOF-CONTRAT,EOF-CREANCIER-CEC
     ELSE IF NU-ID-CRE OF OP-CEC NOT =  NU-ID-CRE OF W-ENTETE-CREANCIER
             MOVE 1 TO EOF-CREANCIER-CEC, EOF-CONTRAT
          ELSE IF NU-DOMI OF OP-CEC  NOT =  NU-DOMI OF W-ENTETE-CONTRAT
                  MOVE 1 TO EOF-CONTRAT.
*
 P-GPE-CONTRAT-CEC-DEST-CEC.
     MOVE 0 TO EOF-CONTRAT.
     MOVE NU-DOMI OF OP-CEC TO NU-DOMI OF W-ENTETE-CONTRAT.
     PERFORM P-OPERATION-CEC-DEST-CEC UNTIL EOF-CONTRAT = 1.
*
```

```
P-OPERATION-CEC-DEST-CEC.
     READ FICH-IN RECORD AT END PERFORM JAMAIS-EXECUTE.
     IF ID-ENREG OF ENREG-FICH-IN  =  9
        MOVE 1 TO EOF-CONTRAT,EOF-CREANCIER-CEC
     ELSE IF NU-ID-CRE OF OP-CEC NOT = NU-ID-CRE OF W-ENTETE-CREANCIER
             MOVE 1 TO EOF-CONTRAT,EOF-CREANCIER-CEC
          ELSE IF NU-DOMI OF OP-CEC  NOT =  NU-DOMI OF W-ENTETE-CONTRAT
                  MOVE 1 TO EOF-CONTRAT.
*
*
 P-GROUPE-CREANCIER.
     MOVE CORR DEBUT-CRE TO W-ENTETE-CREANCIER.
     MOVE  NU-CPTE-CRE OF DEBUT-CRE TO
           NU-CPTE-CRE OF W-OPERATION.
     MOVE 0 TO ENTETE-IMPRIMEE.
     PERFORM P-GROUPE-CONTRAT-CRE
   UNTIL ID-ENREG OF ENREG-FICH-IN  = 9.
     READ FICH-IN RECORD AT END MOVE 1 TO EOF-FICH-IN.
*
 P-GROUPE-CONTRAT-CRE.
     IF DEBUT-NU-DOMI OF OP-CRE = 048
   PERFORM P-GPE-CONTRAT-CRE-DEST-CGER
     ELSE PERFORM P-GPE-CONTRAT-CRE-DEST-CEC.
*
 P-GPE-CONTRAT-CRE-DEST-CGER.
     IF ENTETE-IMPRIMEE = 0  WRITE ENTETE-CREANCIER FROM W-ENTETE-CREANCIER
                             MOVE 1 TO ENTETE-IMPRIMEE.
     MOVE 0 TO EOF-CONTRAT.
     MOVE NU-DOMI OF OP-CRE TO NU-DOMI OF W-ENTETE-CONTRAT.
     WRITE ENTETE-CONTRAT FROM W-ENTETE-CONTRAT.
     PERFORM P-OPERATION-CRE-DEST-CGER UNTIL EOF-CONTRAT = 1.
*
 P-OPERATION-CRE-DEST-CGER.
     MOVE CORR OP-CRE TO W-OPERATION.
     WRITE OPERATION FROM W-OPERATION.
     READ FICH-IN RECORD AT END PERFORM JAMAIS-EXECUTE.
     IF ID-ENREG OF ENREG-FICH-IN = 9
        MOVE 1 TO EOF-CONTRAT
     ELSE IF NU-DOMI OF OP-CRE  NOT =  NU-DOMI OF W-ENTETE-CONTRAT
              MOVE 1 TO EOF-CONTRAT.
*
 P-GPE-CONTRAT-CRE-DEST-CEC.
     MOVE 0 TO EOF-CONTRAT.
     MOVE NU-DOMI OF OP-CRE TO NU-DOMI OF W-ENTETE-CONTRAT.
     PERFORM P-OPERATION-CRE-DEST-CEC UNTIL EOF-CONTRAT = 1.
*
 P-OPERATION-CRE-DEST-CEC.
     READ FICH-IN RECORD AT END PERFORM JAMAIS-EXECUTE.
     IF ID-ENREG OF ENREG-FICH-IN = 9
         MOVE 1 TO EOF-CONTRAT
     ELSE IF NU-DOMI OF OP-CRE  NOT =  NU-DOMI OF W-ENTETE-CONTRAT
             MOVE 1 TO  EOF-CONTRAT.
*
 JAMAIS-EXECUTE.
*
* Ce paragraphe a ete cree parce que la syntaxe COBOL du read impose
* l'existence d'une instruction imperative AT END qui sera executee
* lorsqu'on arrive en fin de fichier.
* Si le fichier d'entree est conforme a la structure donee a la page 7
* l'effet du READ a cet endroit du programme ne provoquera jamais
```

```
* L'exécution de l'instruction impérative puisque conformément à la
* structure du fichier d'entrée on ne devrait jamais être en fin de
* fichier à cet endroit.
  EXIT.
```

A N N E X E 3

```
IDENTIFICATION DIVISION.
PROGRAM-ID. P2.
*
*
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT F2    ASSIGN TO DSK
                 RECORDING MODE IS ASCII
                 ORGANIZATION IS SEQUENTIAL.
    SELECT FC-ACC ASSIGN TO DSK
                  RECORDING MODE IS ASCII
                  ORGANIZATION IS SEQUENTIAL.
    SELECT FC-NONACC ASSIGN TO DSK
                     RECORDING MODE IS ASCII
                     ORGANIZATION IS SEQUENTIAL.
*
*
DATA DIVISION.
*
*
FILE SECTION.
FD F2; LABEL RECORDS ARE STANDARD
       VALUE OF ID IS NOM-EXT-F2.
01  ENTETE-CREANCIER.
    02  ID-ENREG                          PIC 9.
    02  DATE-PIVOT                        PIC 9(6).
    02  PROVENANCE                        PIC 9(11).
    02  NU-ID-CRE                         PIC 9(11).
01  ENTETE-CONTRAT.
    02  ID-ENREG                          PIC 9.
    02  NU-DOMI                           PIC 9(12).
01  OPERATION.
    02  ID-ENREG                          PIC 9.
    02  NATURE-OP                         PIC 9.
    02  MONTANT-OP                        PIC 9(12).
    02  NOM-CRE                           PIC X(26).
    02  COMM1                             PIC X(15).
    02  COMM2                             PIC X(15).
    02  NU-REFERENCE                      PIC 9(12).
    02  NU-CPTE-CRE                       PIC 9(12).
01  ENREG-F2.
    02  ID-ENREG                          PIC 9.
*
FD FC-ACC; LABEL RECORDS ARE STANDARD
           VALUE OF ID IS NOM-EXT-FC-ACC.
01  ENREG-ACC.
    02  DATE-PIVOT                        PIC 9(6).
    02  PROVENANCE                        PIC 9(11).
    02  NU-ID-CRE                         PIC 9(11).
    02  NU-CPTE-CRE                       PIC 9(12).
    02  NU-DOMI                           PIC 9(12).
    02  NATURE-OP                         PIC 9.
    02  MONTANT-OP                        PIC 9(12).
    02  NOM-CRE                           PIC X(26).
    02  COMM1                             PIC X(15).
    02  COMM2                             PIC X(15).
    02  NU-REFERENCE                      PIC 9(12).
    02  DEB-NU-CPTE                       PIC 9(12).
    02  ID-GESTION                        PIC X(5).
```

```
*
FD FC-NOMACC; LABEL RECORDS ARE STANDARD
                VALUE OF ID IS NOM-EXT-FC-NOMACC.
01  ENREG-NOMACC.
    02  DATE-PIVOT                          PIC 9(6).
    02  PROVENANCE                          PIC 9(11).
    02  NU-ID-CRE                           PIC 9(11).
    02  NU-CPTE-CRE                         PIC 9(12).
    02  NU-DOM1                             PIC 9(12).
    02  NATURE-OP                           PIC 9.
    02  MONTANT-OP                          PIC 9(12).
    02  NOM-CRE                             PIC X(26).
    02  COMM1                               PIC X(15).
    02  COMM2                               PIC X(15).
    02  NU-REFERENCE                        PIC 9(12).
    02  DEB-NU-CPTE                         PIC 9(12).
    02  ID-GESTION                          PIC X(5).
    02  TABLEAU                             PIC 9(7).
    02  INC-NU-ID-CRE                       PIC 9(11).
    02  INC-NU-REF                          PIC 9(12).
    02  DATE-REVOCATION                     PIC 9(6).
    02  DESC-OP-CRE                         PIC X(30).
    02  DESC-OP-CDD                         PIC X(30).
*
WORKING-STORAGE SECTION.
01  EOF-F2                                  PIC 9.
01  EOF-CREANCIER                           PIC 9.
01  EOF-CONTRAT                             PIC 9.
01  CREEXIST                                PIC 9.
01  OPPCREANCIER                            PIC 9.
01  W-DESC-OP-CRE                           PIC X(30).
01  CDDEXIST                                PIC 9.
01  IDENTCRE                                PIC 9(11).
01  NUREF                                   PIC 9(12).
01  OPPCDD                                  PIC 9.
01  ETATCDD                                 PIC 9.
01  W-DATE-REVOCATION                       PIC 9(6).
01  W-DESC-OP-CDD                           PIC X(30).
01  W-NU-DOM1                               PIC 9(12).
01  W-NU-ID-CRE                             PIC 9(11).
01  W-ENREG.
    02  DATE-PIVOT                          PIC 9(6).
    02  PROVENANCE                          PIC 9(11).
    02  NU-ID-CRE                           PIC 9(11).
    02  NU-DOM1                             PIC 9(12).
    02  DEB-NU-CPTE                         PIC 9(12).
    02  ID-GESTION                          PIC X(5).
    02  TABLEAU                             PIC 9(7).
    02  TAB REDEFINES TABLEAU.
        03  ELE  OCCURS  7  TIMES  PIC 9.
    02  INC-NU-ID-CRE                       PIC 9(11).
    02  INC-NU-REF                          PIC 9(12).
    02  DATE-REVOCATION                     PIC 9(6).
    02  DESC-OP-CRE                         PIC X(30).
    02  DESC-OP-CDD                         PIC X(30).
01  CPTE-DEB.
    02  D-CPTE-DEB                          PIC 9(3).
    02  C-CPTE-DEB.
        03  C-DEUX                          PIC 9(2).
        03  C-CTNO                          PIC 9(5).
```

```
        88  INTERVALLE     VALUES  ARE  0052000 THRU 0052999.
    02  E-CPTE-DEB                          PIC 9(2).
 77 NOM-EXT-F2                              PIC X(9).
 77 NOM-EXT-FO-ACC                          PIC X(9).
 77 NOM-EXT-FO-NONACC                       PIC X(9).
*
*
 PROCEDURE DIVISION.
*
 PROCESS-F2.
     PERFORM  LIRE-NOM-FICH.
     OPEN INPUT F2.
     OPEN OUTPUT FO-ACC, FO-NONACC.
     MOVE 0 TO EOF-F2.
     READ F2 RECORD AT END MOVE 1 TO EOF-F2.
     PERFORM  P-GROUPE-CREANCIER  UNTIL  EOF-F2 = 1.
     CLOSE F2, FO-ACC, FO-NONACC.
     STOP RUN.
*
 LIRE-NOM-FICH.
     DISPLAY  "VEUILLEZ DONNER LES NOMS EXTERNES DES FICHIERS".
     DISPLAY  "MAX 9 CHAR : FFFFFFSSS DEVIENDRA FFFFFF.SSS".
     DISPLAY  "FICHIER F2  : " WITH NO ADVANCING.
     ACCEPT  NOM-EXT-F2.
     DISPLAY  "FICHIER FO-ACC : " WITH NO ADVANCING.
     ACCEPT  NOM-EXT-FO-ACC.
     DISPLAY  "FICHIER FO-NONACC : " WITH NO ADVANCING.
     ACCEPT  NOM-EXT-FO-NONACC.
*
 P-GROUPE-CREANCIER.
     MOVE 0 TO EOF-CREANCIER.
     MOVE CORR ENTETE-CREANCIER  TO  W-ENREG.
     MOVE NU-ID-CRE OF ENTETE-CREANCIER TO W-NU-ID-CRE.
     CALL  EXICRE  USING   W-NU-ID-CRE,
                           CREEXIST, OPPCREANCIER,W-DESC-OP-CRE.
     MOVE SPACE TO DESC-OP-CRE OF W-ENREG.
     MOVE ZERO TO ELE OF W-ENREG (1), ELE OF W-ENREG (2).
     IF CREEXIST = 0   MOVE 1 TO ELE OF W-ENREG (1)
        ELSE IF OPPCREANCIER = 1  MOVE 1 TO ELE OF W-ENREG (2)
                                  MOVE W-DESC-OP-CRE TO DESC-OP-CRE OF W-ENREG
     READ F2 RECORD AT END MOVE 1 TO EOF-F2.
     IF EOF-F2 = 1 OR  ID-ENREG OF ENREG-F2 = 0
                       MOVE 1 TO EOF-CREANCIER.
     PERFORM  P-GROUPE-CONTRAT  UNTIL  EOF-CREANCIER = 1.
*
 P-GROUPE-CONTRAT.
     MOVE 0 TO EOF-CONTRAT.
     MOVE NU-DOMI OF ENTETE-CONTRAT  TO  NU-DOMI OF W-ENREG,
                                         W-NU-DOMI.
     CALL  EXICDD USING  W-NU-DOMI, CDDEXIST,
                         IDENTCRE, NUREF, OPPCDD, ETATCDD,
                         W-DATE-REVOCATION,
                         W-DESC-OP-CDD,CPTE-DEB.
     MOVE ALL ZERO TO DATE-REVOCATION OF W-ENREG.
     MOVE ALL SPACE TO DESC-OP-CDD OF W-ENREG.
     MOVE ZERO TO ELE OF W-ENREG (3), ELE OF W-ENREG (4),
                  ELE OF W-ENREG (5), ELE OF W-ENREG (6).
     IF CDDEXIST = 0  MOVE 1 TO ELE  OF W-ENREG (3)
                      MOVE SPACE TO DEB-NU-CPTE OF W-ENREG,
                                    ID-GESTION  OF W-ENREG
```

```
      ELSE  PERFORM PROCESS-DEDUCT
            PERFORM PROCESS-CDD-EXISTANT.
    READ F2 RECORD AT END MOVE 1 TO EOF-F2.
    IF EOF-F2 = 1  OR  ID-ENREG OF ENREG-F2 = 0
            MOVE 1 TO EOF-CREANCIER,EOF-CONTRAT
      ELSE IF ID-ENREG OF ENREG-F2 = 2
            MOVE 1 TO EOF-CONTRAT.
    PERFORM  P-OPERATION  UNTIL  EOF-CONTRAT = 1.
*
 PROCESS-CDD-EXISTANT.
            IF OPPCDD = 1  MOVE 1 TO ELE OF W-ENREG (5)
                           MOVE W-DESC-OP-CDD TO DESC-OP-CDD OF W-ENREG
               ELSE NEXT SENTENCE.
            IF ETATCDD = 1  MOVE 1 TO ELE OF W-ENREG (4)
                            MOVE W-DATE-REVOCATION
                                  TO DATE-REVOCATION OF W-ENREG
               ELSE NEXT SENTENCE.
            IF IDENTCRE  NOT =  NU-ID-CRE OF W-ENREG
               MOVE 1 TO ELE  OF W-ENREG (6)
               MOVE IDENTCRE TO INC-NU-ID-CRE OF W-ENREG.
*
 P-OPERATION.
    MOVE ZERO TO ELE OF W-ENREG (7).
    IF NUREF  NOT =  NU-REFERENCE OF OPERATION
            MOVE 1 TO ELE OF W-ENREG (7)
            MOVE NUREF TO INC-NU-REF OF W-ENREG.
    IF TAB OF W-ENREG  =  ALL ZERO
            MOVE CORR W-ENREG TO ENREG-ACC
            MOVE CORR OPERATION TO ENREG-ACC
            WRITE ENREG-ACC
      ELSE  MOVE CORR W-ENREG TO ENREG-NONACC
            MOVE CORR OPERATION TO ENREG-NONACC
            WRITE ENREG-NONACC.
    READ F2 RECORD AT END MOVE 1 TO EOF-F2.
    IF EOF-F2 = 1    MOVE 1 TO EOF-CONTRAT, EOF-CREANCIER
      ELSE  IF ID-ENREG OF ENREG-F2 = 0
                  MOVE 1 TO EOF-CONTRAT, EOF-CREANCIER
               ELSE IF ID-ENREG OF ENREG-F2 = 2
                  MOVE 1 TO EOF-CONTRAT.
*
 PROCESS-DEDUCT.
    MOVE CPTE-DEB  TO  DEB-NU-CPTE OF W-ENREG.
    IF D-CPTE-DEB = 001 AND C-CPTE-DEB NOT < 50000
                        AND NOT INTERVALLE
           MOVE '001N ' TO ID-GESTION OF W-ENREG
    ELSE IF D-CPTE-DEB =001 AND C-CPTE-DEB <50000
                        AND INTERVALLE
              MOVE '001F ' TO ID-GESTION OF W-ENREG
         ELSE IF D-CPTE-DEB = 004 AND C-DEUX = 25
                  MOVE '00425' TO ID-GESTION OF W-ENREG
              ELSE IF D-CPTE-DEB = 011 AND C-DEUX = 25
                        MOVE '01125' TO ID-GESTION OF W-ENREG
                  ELSE IF D-CPTE-DEB = 034
                              MOVE '034N ' TO ID-GESTION OF W-ENREG
                        ELSE IF D-CPTE-DEB = 035
                                    MOVE '035F ' TO ID-GESTION OF W-ENREG
                              ELSE
                              MOVE D-CPTE-DEB TO ID-GESTION OF W-ENREG.
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXICRE.
*
*
DATA DIVISION.
*
LINKAGE SECTION.
01  IDENT-CREANCIER                    PIC 9(11).
01  CREANCIER-EXISTE                   PIC 9.
01  OPPOSITION-CREANCIER               PIC 9.
01  DESC-OP-CREANCIER   PIC X(30).
*
*
PROCEDURE DIVISION.
*
ENTRY 'EXICRE' USING  IDENT-CREANCIER, CREANCIER-EXISTE, OPPOSITION-CREANCIER
                      DESC-OP-CREANCIER.
*
DISPLAY "Pour ce numero d identification de creancier : " WITH NO ADVANCING.
DISPLAY IDENT-CREANCIER.
DISPLAY "Taper 1 si il existe,0 sinon suivi de <return>".
ACCEPT  CREANCIER-EXISTE.
IF  CREANCIER-EXISTE = 1
    DISPLAY "Taper 1 si il y a une opposition sur ce creancier "
    DISPLAY " 0 sinon suivi de <return>"
    ACCEPT  OPPOSITION-CREANCIER
    IF OPPOSITION-CREANCIER = 1
       DISPLAY "Donnez la description de l opposition"
       ACCEPT DESC-OP-CREANCIER.
EXIT PROGRAM.
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXICDD.
*
*
DATA DIVISION.
*
LINKAGE SECTION.
*
01  NU-DOMI                    PIC 9(12).
01  CDDEXIST                   PIC 9.
01  IDENTCRE                   PIC 9(11).
01  NUREF                      PIC 9(12).
01  OPPCDD                     PIC 9.
01  ETATCDD                    PIC 9.
01  DATE-REVOCATION            PIC 9(6).
01  DESC-OP-CDD                PIC X(30).
01  CPTE-DEB                   PIC 9(12).
*
*
PROCEDURE DIVISION.
*
ENTRY  'EXICDD'  USING NU-DOMI, CDDEXIST, IDENTCRE, NUREF, OPPCDD,
                       ETATCDD, DATE-REVOCATION,DESC-OP-CDD, CPTE-DEB.
*
DISPLAY "Pour ce numero de domiciliation : " WITH NO ADVANCING.
DISPLAY  NU-DOMI.
DISPLAY "Taper 1 si il existe,0 sinon suivi de <return> :".
ACCEPT  CDDEXIST.
IF  CDDEXIST  = 1
    DISPLAY "Donnez le numero d identification du creancier : "
    ACCEPT  IDENTCRE
    DISPLAY "Donnez le numero de reference : "
    ACCEPT  NUREF
    DISPLAY "Taper 1 si il y a une opposition sur ce contrat,0 sinon : "
    ACCEPT  OPPCDD
    PERFORM TRT-OPPOSITION
    DISPLAY "Taper 0 si l etat du contrat est en cours, 1 si revoque : "
    ACCEPT  ETATCDD
    PERFORM TRT-ETATCDD
    DISPLAY "Donnez le numero de cpte debiteur se trouvant sur ce contrat :
    ACCEPT  CPTE-DEB.
EXIT PROGRAM.
*
TRT-OPPOSITION.
    IF OPPCDD = 1
       DISPLAY "Donnez la description de l opposition"
       ACCEPT DESC-OP-CDD.
*
TRT-ETATCDD.
    IF ETATCDD = 1
       DISPLAY "Donner la date de revocation"
       ACCEPT DATE-REVOCATION.
```

A N N E X E 4

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. PW1.
*
*
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT FICH-IN ASSIGN TO DSK
                    RECORDING MODE IS ASCII
                    ORGANIZATION IS SEQUENTIAL.
*
*
 DATA DIVISION.
*
 FILE SECTION.
 FD  FICH-IN; LABEL RECORDS ARE STANDARD
              VALUE OF ID IS  NOM-EXT-FICH-IN.
 01  DEBUT-CEC.
     02  ID-ENREG                    PIC 9.
     02  DATE-PIVOT                  PIC 9(6).
     02  PROVENANCE                  PIC 9(3).
 01  DEBUT-CRE.
     02  ID-ENREG                    PIC 9.
     02  FILLER                      PIC X(12).
     02  DATE-PIVOT                  PIC 9(6).
     02  FILLER                      PIC X(15).
     02  PROVENANCE                  PIC 9(11).
     02  NU-ID-CRE                   PIC 9(11).
     02  NU-CPTE-CRE                 PIC 9(12).
 01  OP-CEC.
     02  ID-ENREG                    PIC 9.
     02  FILLER                      PIC X(21).
     02  NU-DOMI.
         03  DEBUT-NU-DOMI            PIC 9(3).
         03  FIN-NU-DOMI              PIC 9(9).
     02  NU-CPTE-CRE                 PIC 9(12).
     02  NATURE-OP                   PIC 9.
     02  MONTANT-OP                  PIC 9(12).
     02  NOM-CRE                     PIC X(26).
     02  NU-ID-CRE                   PIC 9(11).
     02  COMM1                       PIC X(15).
     02  COMM2                       PIC X(15).
     02  NU-REFERENCE                PIC 9(12).
 01  OP-CRE.
     02  ID-ENREG                    PIC 9.
     02  FILLER                      PIC X(6).
     02  NU-DOMI.
         03  DEBUT-NU-DOMI           PIC 9(3).
         03  FIN-NU-DOMI             PIC 9(9).
     02  NATURE-OP                   PIC 9.
     02  MONTANT-OP                  PIC 9(12).
     02  NOM-CRE                     PIC X(26).
     02  COMM1                       PIC X(15).
     02  COMM2                       PIC X(15).
     02  NU-REFERENCE                PIC 9(12).
 01  ENREG-FICH-IN.
     02 ID-ENREG                     PIC 9.
*
 WORKING-STORAGE SECTION.
 01  EOF-CREANCIER-CEC               PIC 9.
```

```
 01  EOF-CONTRAT                          PIC 9.
 01  ENTEETE-TRPETRPE                     PIC 9.
 01  EOF-FICH-IN                          PIC 9.
 01  QS                                   PIC 99 VALUE 1.
 77  NOM-EXT-FICH-IN                      PIC X(9).
 01  W-ENTETE-CREANCIER.
     02  ID-ENREG       PIC 9 VALUE IS 0.
     02  DATE-PIVOT     PIC 9(6).
     02  PROVENANCE     PIC 9(11).
     02  NU-ID-CRE      PIC 9(11).
 01  W-ENTETE-CONTRAT.
     02  ID-ENREG       PIC 9  VALUE IS 2.
     02  NU-DOMI        PIC 9(12).
 01  W-OPERATION.
     02  ID-ENREG       PIC 9  VALUE IS 1.
     02  NATURE-OP      PIC 9.
     02  MONTANT-OP     PIC 9(12).
     02  NOM-CRE        PIC X(26).
     02  COMM1 PIC X(15).
     02  COMM2 PIC X(15).
     02  NU-REFERENCE PIC 9(12).
     02  NU-CPTE-CRE  PIC 9(12).
*
 LINKAGE SECTION.
 01  EOF-F2                              PIC 9.
 01  F2-RECORD.
     02  ENTETE-CREANCIER.
         03  ID-ENREG                        PIC 9.
         03  DATE-PIVOT                      PIC 9(6).
         03  PROVENANCE                      PIC 9(11).
         03  NU-ID-CRE                       PIC 9(11).
         03  FILLER                          PIC X(65).
     02  ENTETE-CONTRAT REDEFINES ENTETE-CREANCIER.
         03  ID-ENREG                        PIC 9.
         03  NU-DOMI                         PIC 9(12).
         03  FILLER                          PIC X(81).
     02  OPERATION REDEFINES ENTETE-CREANCIER.
         03  ID-ENREG                        PIC 9.
         03  NATURE-OP                       PIC 9.
         03  MONTANT-OP                      PIC 9(12).
         03  NOM-CRE                         PIC X(26).
         03  COMM1                           PIC X(15).
         03  COMM2                           PIC X(15).
         03  NU-REFERENCE                    PIC 9(12).
         03  NU-CPTE-CRE                     PIC 9(12).
     02  ENREG-F2 REDEFINES ENTETE-CREANCIER.
         03  ID-ENREG                        PIC 9.
         03  FILLER                          PIC X(93).
*
*
 PROCEDURE DIVISION.
*
 ENTRY 'PW1' USING F2-RECORD, EOF-F2.
 GO TO Q1, Q2, Q3, Q4, Q5, Q6, Q7,Q8  DEPENDING ON QS.
*
 PROCESS-FICH-IN.
 Q1.
     DISPLAY  "VEUILLEZ DONNER LES NOMS EXTERNES DES FICHIERS".
     DISPLAY  "MAX 9 CHAR : FFFFFFSSS DEVIENDRA FFFFFF.SSS".
     DISPLAY  "FICHIER FICH-IN  : " WITH NO ADVANCING.
```

```
     ACCEPT  NOM-EXE-FICH-IN.
     MOVE 0 TO EOF-F2.
     MOVE 0 TO EOF-FICH-IN.
     OPEN INPUT FICH-IN.
     READ FICH-IN RECORD AT END MOVE 1 TO EOF-FICH-IN.
*
 PROCESS-BODY.
     IF EOF-FICH-IN = 1  GO TO PROCESS-FIN.
*
 PROCESS-GROUPE.
     IF PROVENANCE OF DEBUT-CEC NOT = 102  GO TO  P-GROUPE-CRE.
*
 P-GROUPE-CEC.
     MOVE CORR DEBUT-CEC TO W-ENTETE-CREANCIER.
     READ FICH-IN RECORD AT END PERFORM JAMAIS-EXECUTE.
*
 P-BODY-GROUPE-CEC.
     IF ID-ENREG OF ENREG-FICH-IN  =  9  GO TO P-FIN-GROUPE-CEC.
*
 P-GROUPE-CREANCIER-CEC.
     MOVE 0 TO EOF-CREANCIER-CEC.
     MOVE NU-ID-CRE OF OP-CEC  TO  NU-ID-CRE OF W-ENTETE-CREANCIER.
     MOVE 0 TO ENTETE-IMPRIMEE.
*
 P-BODY-CREANCIER-CEC.
     IF EOF-CREANCIER-CEC = 1 GO TO  P-FIN-GPE-CREANCIER-CEC.
*
 P-GROUPE-CONTRAT-CEC.
     IF DEBUT-NU-DOMI OF OP-CEC  NOT =  048 GO TO  P-GPE-CONTRAT-CEC-DEST-CEC.
*
*-----------------------------------------------------------------
*
 P-GPE-CONTRAT-CEC-DEST-CGER.
     IF ENTETE-IMPRIMEE = 0
        MOVE CORR W-ENTETE-CREANCIER  TO  ENTETE-CREANCIER
        MOVE 1 TO ENTETE-IMPRIMEE
        MOVE 0 TO EOF-F2
        MOVE 2 TO QS
        GO TO PTERMIN.
*
 Q2.
*
     MOVE 0 TO EOF-CONTRAT.
     MOVE NU-DOMI OF OP-CEC TO  NU-DOMI OF W-ENTETE-CONTRAT.
     MOVE CORR W-ENTETE-CONTRAT TO  ENTETE-CONTRAT.
     MOVE 0 TO EOF-F2.
     MOVE 3 TO QS.
     GO TO PTERMIN.
*
 Q3.
*
 P-BODY-OPERATION-CEC-DEST-CGER.
     IF EOF-CONTRAT = 1  GO TO P-FIN-GPE-OP-CEC-DEST-CGER.
*
 P-OPERATION-CEC-DEST-CGER.
     MOVE CORR OP-CEC TO W-OPERATION.
     MOVE CORR W-OPERATION TO OPERATION.
     MOVE 0 TO EOF-F2.
     MOVE 4 TO QS.
     GO TO PTERMIN.
```

```
*
 34.
*
     READ FICH-IN RECORD AT END PERFORM  JAMAIS-EXECUTE.
     IF ID-ENREG OF ENREG-FICH-IN = 9 OR NU-ID-CRE OF OP-CEC
                                   NOT = NU-ID-CRE OF W-ENTETE-CREANCIER
          MOVE 1 TO EOF-CONTRAT,EOF-CREANCIER-CEC
     ELSE IF NU-DOMI OF OP-CEC  NOT =  NU-DOMI OF W-ENTETE-CONTRAT
                    MOVE 1 TO EOF-CONTRAT.
*
     GO TO P-BODY-OPERATION-CEC-DEST-CGEP.
*
 P-FIN-GPE-OP-CEC-DEST-CGEP.
     GO TO P-BODY-CREANCIER-CEC.
*
*-------------------------------------------------------
*
 P-GPE-CONTRAT-CEC-DEST-CEC.
*
     MOVE 0 TO EOF-CONTRAT.
     MOVE NU-DOMI OF OP-CEC TO NU-DOMI OF W-ENTETE-CONTRAT.
*
 P-BODY-OPERATION-CEC-DEST-CEC.
     IF EOF-CONTRAT = 1  GO TO P-FIN-GPE-OP-CEC-DEST-CEC.
     READ FICH-IN RECORD AT END PERFORM  JAMAIS-EXECUTE.
     IF ID-ENREG OF ENREG-FICH-IN  =  9 OR NU-ID-CRE OF OP-CEC
                                  NOT = NU-ID-CRE OF W-ENTETE-CREANCIER
          MOVE 1 TO EOF-CONTRAT,EOF-CREANCIER-CEC
     ELSE IF NU-DOMI OF OP-CEC  NOT =  NU-DOMI OF W-ENTETE-CONTRAT
                    MOVE 1 TO EOF-CONTRAT.
     GO TO P-BODY-OPERATION-CEC-DEST-CEC.
*
 P-FIN-GPE-OP-CEC-DEST-CEC.
      GO TO P-BODY-CREANCIER-CEC.
*
*-----------------------------------------------------------
*
 P-FIN-GPE-CREANCIER-CEC.
     GO TO P-BODY-GROUPE-CEC.
*
 P-FIN-GROUPE-CEC.
     READ FICH-IN RECORD AT END MOVE 1 TO EOF-FICH-IN.
*
 GO TO PROCESS-FIN-GROUPE.
*
*****************************************************************
*
 P-GROUPE-CRE.
     MOVE CORR DEBUT-CRE TO W-ENTETE-CREANCIER.
     MOVE NU-CPTE-CRE OF DEBUT-CRE  TO  NU-CPTE-CRE OF W-OPERATION.
     MOVE 0 TO ENTETE-IMPRIMEE.
*
 P-BODY-GROUPE-CRE.
     IF ID-ENREG OF ENREG-FICH-IN = 9  GO TO  P-FIN-GROUPE-CRE.
*
 P-GROUPE-CONTRAT-CRE.
     IF DEBUT-NU-DOMI OF OP-CRE NOT = 048  GO TO P-GPE-CONTRAT-CRE-DEST-CEC.
*
 P-GPE-CONTRAT-CRE-DEST-CGER.
     IF ENTETE-IMPRIMEE = 0
```

```
        MOVE CORR W-ENTETE-CREANCIER TO ENTETE-CREANCIER
        MOVE 1 TO ENTETE-IMPRIMER
        MOVE 0 TO EOF-F2
        MOVE 5 TO QS
        GO TO PTERMIN.
*
 G5.
*
     MOVE 0 TO EOF-CONTRAT.
     MOVE NU-DOMI OF OP-CRE TO  NU-DOMI OF W-ENTETE-CONTRAT.
     MOVE CORR W-ENTETE-CONTRAT TO ENTETE-CONTRAT.
     MOVE 0 TO EOF-F2.
     MOVE 6 TO QS.
     GO TO PTERMIN.
*
 G6.
*
 P-BODY-OPERATION-CRE-DEST-CGER.
     IF EOF-CONTRAT = 1  GO TO  P-FIN-GPE-OP-CRE-DEST-CGER.
*
 P-OPERATION-CRE-DEST-CGER.
     MOVE CORR OP-CRE TO W-OPERATION.
     MOVE CORR W-OPERATION  TO  OPERATION.
     MOVE 0 TO EOF-F2.
     MOVE 7 TO QS.
     GO TO PTERMIN.
*
 G7.
     READ FICH-IN RECORD AT END PERFORM JAMAIS-EXECUTE.
     IF ID-ENREG OF ENREG-FICH-IN = 9 OR NU-DOMI OF OP-CRE
                                 NOT = NU-DOMI OF W-ENTETE-CONTRAT
          MOVE 1 TO EOF-CONTRAT.

*
     GO TO P-BODY-OPERATION-CRE-DEST-CGER.
*
 P-FIN-GPE-OP-CRE-DEST-CGER.
     GO TO P-BODY-GROUPE-CRE.
*
*
 P-GPE-CONTRAT-CRE-DEST-CEC.
     MOVE 0 TO EOF-CONTRAT.
     MOVE NU-DOMI OF OP-CRE  TO  NU-DOMI OF W-ENTETE-CONTRAT.
*
 P-BODY-OPERATION-CRE-DEST-CEC.
     IF EOF-CONTRAT = 1  GO TO  P-FIN-GPE-OP-CRE-DEST-CEC.
*
 P-OPERATION-CRE-DEST-CEC.
     READ FICH-IN RECORD AT END PERFORM JAMAIS-EXECUTE.
     IF ID-ENREG OF ENREG-FICH-IN = 9 OR NU-DOMI OF OP-CRE
                                 NOT =  NU-DOMI OF W-ENTETE-CONTRAT
          MOVE 1 TO EOF-CONTRAT.
*
     GO TO P-BODY-OPERATION-CRE-DEST-CGER.
*
 P-FIN-GPE-OP-CRE-DEST-CEC.
      GO TO P-BODY-GROUPE-CRE.
*
*
 P-FIN-GROUPE-CRE.
```

```
     READ FICH-IN RECORD AT END MOVE 1 TO EOF-FICH-IN.
     GO TO PROCESS-FIN-GROUPE.
*
*****************************************************************
*
 PROCESS-FIN-GROUPE.
  GO TO PROCESS-BODY.
*
 PROCESS-FIN.
     CLOSE FICH-IN.
     MOVE 1 TO EOF-F2.
     MOVE 8 TO QS.
     GO TO PTERMIN.
*
 Q8.
*
 PTERMIN.
     GOBACK.
*
 JAMAIS-EXECUTE.
*
* Ce paragraphe a ete cree parce que la syntaxe COBOL du read impose
* l'existence d'une instruction imperative AT END qui sera executee
* lorsqu'on arrive en fin de fichier.
* Si le fichier d'entree est conforme a la structure donnee a la page 2
* l'effet du READ a cet endroit du programme ne provoquera jamais
* l'execution de l'instruction imperative puisqu'on ne devrait jamais
* etre en fin de fichier.
  EXIT.
```

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. PW2.
*
*
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT FO-ACC ASSIGN TO DSK
                   RECORDING MODE IS ASCII
                   ORGANIZATION IS SEQUENTIAL.
     SELECT FO-NONACC ASSIGN TO DSK
                      RECORDING MODE IS ASCII
                      ORGANIZATION IS SEQUENTIAL.
*
*
 DATA DIVISION.
*
*
 FILE SECTION.
*
 FD FO-ACC; LABEL RECORDS ARE STANDARD
            VALUE OF ID IS NOM-EXT-FO-ACC.
 01  ENREG-ACC.
     02  DATE-PIVOT                          PIC 9(6).
     02  PROVENANCE                          PIC 9(11).
     02  NU-ID-CRE                           PIC 9(11).
     02  NU-CPTE-CRE                         PIC 9(12).
     02  NU-DOMI                             PIC 9(12).
     02  NATURE-OP                           PIC 9.
     02  MONTANT-OP                          PIC 9(12).
     02  NOM-CPE                             PIC X(26).
     02  COMM1                               PIC X(15).
     02  COMM2                               PIC X(15).
     02  NU-REFERENCE                        PIC 9(12).
     02  DEB-NU-CPTE                         PIC 9(12).
     02  ID-GESTION                          PIC X(5).
*
 FD FO-NONACC; LABEL RECORDS ARE STANDARD
               VALUE OF ID IS NOM-EXT-FO-NONACC.
 01  ENREG-NONACC.
     02  DATE-PIVOT                          PIC 9(6).
     02  PROVENANCE                          PIC 9(11).
     02  NU-ID-CRE                           PIC 9(11).
     02  NU-CPTE-CRE                         PIC 9(12).
     02  NU-DOMI                             PIC 9(12).
     02  NATURE-OP                           PIC 9.
     02  MONTANT-OP                          PIC 9(12).
     02  NOM-CPE                             PIC X(26).
     02  COMM1                               PIC X(15).
     02  COMM2                               PIC X(15).
     02  NU-REFERENCE                        PIC 9(12).
     02  DEB-NU-CPTE                         PIC 9(12).
     02  ID-GESTION                          PIC X(5).
     02  TABLEAU                             PIC 9(7).
     02  INC-NU-ID-CRE                       PIC 9(11).
     02  INC-NU-REF                          PIC 9(12).
     02  DATE-REVOCATION                     PIC 9(6).
     02  DESC-OP-CRE                         PIC X(30).
     02  DESC-OP-CDB                         PIC X(30).
*
```

```
WORKING-STORAGE SECTION.
*
 01  LGI-E2                                  PIC 9.
*
 01  E2-RECORD.
     02  ENTETE-CREANCIER.
         03  ID-ENREG                        PIC 9.
         03  DATE-PIVOT                      PIC 9(6).
         03  PROVENANCE                      PIC 9(11).
         03  NU-TD-CRE                       PIC 9(11).
         03  FILLER                          PIC X(65).
     02  ENTETE-CONTRAT REDEFINES ENTETE-CREANCIER.
         03  ID-ENREG                        PIC 9.
         03  NU-DOM1                         PIC 9(12).
         03  FILLER                          PIC X(81).
     02  OPERATION REDEFINES ENTETE-CREANCIER.
         03  ID-ENREG                        PIC 9.
         03  NATURE-OP                       PIC 9.
         03  MONTANT-OP                      PIC 9(12).
         03  NOM-CRE                         PIC X(26).
         03  COMM1                           PIC X(15).
         03  COMM2                           PIC X(15).
         03  NU-REFERENCE                    PIC 9(12).
         03  NU-CPTE-CRE                     PIC 9(12).
     02  ENREG-E2 REDEFINES ENTETE-CREANCIER.
         03  ID-ENREG                        PIC 9.
         03  FILLER                          PIC X(93).
*
 01  EOF-CREANCIER                           PIC 9.
 01  EOF-CONTRAT                             PIC 9.
 01  CREEXIST                                PIC 9.
 01  OPPCREANCIER                            PIC 9.
 01  W-DESC-OP-CRE                           PIC X(30).
 01  CDDEXIST                                PIC 9.
 01  IDENTICRE                               PIC 9(11).
 01  FULFF                                   PIC 9(12).
 01  CPECLD                                  PIC 9.
 01  ETATCDD                                 PIC 9.
 01  W-DATE-REVOCATION                       PIC 9(6).
 01  W-DESC-OP-CDD                           PIC X(30).
 01  W-NU-DOM1                               PIC 9(12).
 01  W-NU-TD-CRE                             PIC 9(11).
 01  W-ENREG.
     02  DATE-PIVOT                          PIC 9(6).
     02  PROVENANCE                          PIC 9(11).
     02  NU-TD-CRE                           PIC 9(11).
     02  NU-CPTE-CRE                         PIC 9(12).
     02  NU-DOM1                             PIC 9(12).
     02  NATURE-OP                           PIC 9.
     02  MONTANT-OP                          PIC 9(12).
     02  NOM-CRE                             PIC X(26).
     02  COMM1                               PIC X(15).
     02  COMM2                               PIC X(15).
     02  NU-REFERENCE                        PIC 9(12).
     02  DEB-NU-CPTE                         PIC 9(12).
     02  ID-GESTION                          PIC X(5).
     02  TABLEAU                             PIC 9(7).
     02  TAB REDEFINES TABLEAU.
         03  ELE  OCCURS  7  TIMES  PIC 9.
     02  INC-NU-ID-CRE                       PIC 9(11).
```

```
     02  IPC-NU-REF                          PIC 9(12).
     02  DATE-REVOCATION                     PIC 9(6).
     02  DESC-OP-CRE                         PIC X(30).
     02  DESC-OP-CDE                         PIC X(30).
 01  CPTE-DEB.
     02  E-CPTE-DEB                          PIC 9(3).
     02  C-CPTE-DEB.
         03  C-LTGX                          PIC 9(2).
         03  C-CTGQ                          PIC 9(5).
         88  INTERVALLE     VALUES  ARE  0052000 THRU 0052999.
     02  F-CPTE-DEB                          PIC 9(2).
 77  NOM-EXT-FO-ACC                          PIC X(9).
 77  NOM-EXT-FO-NONACC                       PIC X(9).
*
*
 PROCEDURE DIVISION.
*
 PROCESS-F2.
     PERFORM  LIRE-NOM-FICH.
     OPEN OUTPUT FO-ACC, FO-NONACC.
     MOVE 0 TO EOF-F2.
     CALL PW1 USING  F2-RECORD, EOF-F2.
     PERFORM  P-GROUPE-CREANCIER  UNTIL  EOF-F2 = 1.
     CLOSE FO-ACC, FO-NONACC.
     CALL PW1 USING  F2-RECORD, EOF-F2.
     STOP RUN.
*
 LIRE-NOM-FICH.
     DISPLAY  "VEUILLEZ DONNER LES NOMS EXTERNES DES FICHIERS".
     DISPLAY  "MAX 9 CHAR : FFFFFFSSS DEVIENDRA FFFFFF.SSS".
     DISPLAY  "FICHIER FO-ACC : " WITH NO ADVANCING.
     ACCEPT  NOM-EXT-FO-ACC.
     DISPLAY  "FICHIER FO-NONACC : " WITH NO ADVANCING.
     ACCEPT  NOM-EXT-FO-NONACC.
*
 P-GROUPE-CREANCIER.
     MOVE 0 TO EOF-CREANCIER.
     MOVE CORR ENTETE-CREANCIER  TO  W-ENREG.
     MOVE NU-ID-CRE OF ENTETE-CREANCIER TO W-NU-ID-CRE.
     CALL  EXICRE  USING  W-NU-ID-CRE
                          CREEXIST, OPPCREANCIER,
                          W-DESC-OP-CRE.
     MOVE SPACE TO DESC-OP-CRE OF W-ENREG.
     MOVE ZERO TO FLE OF W-ENREG (1), FLE OF W-ENREG (2).
     IF CREEXIST = 0        MOVE 1 TO FLE OF W-ENREG (1)
        ELSE IF OPPCREANCIER = 1  MOVE 1 TO FLE OF W-ENREG (2).
     CALL PW1 USING  F2-RECORD, EOF-F2.
     MOVE W-DESC-OP-CRE TO DESC-OP-CRE OF W-ENREG.
     IF EOF-F2 = 1 OR  ID-ENREG OF ENREG-F2 = 0
         MOVE 1 TO EOF-CREANCIER.
     PERFORM  P-GROUPE-CONTRAT  UNTIL  EOF-CREANCIER = 1.
*
 P-GROUPE-CONTRAT.
     MOVE 0 TO EOF-CONTRAT.
     MOVE NU-DOMI OF ENTETE-CONTRAT  TO  W-NU-DOMI,NU-DOMI OF W-ENREG.
     CALL  EXICDD USING  W-NU-DOMI, CDDEXIST,
                         IDENTCRE, NUREF, OPPCDD, ETATCDD,
                         W-DATE-REVOCATION ,
                         W-DESC-OP-CDE ,
                         CPTE-DEB.
```

```
     MOVE ALL ZERO TO DATE-REVOCATION OF W-ENREG.
     MOVE ALL SPACE TO DESC-OP-CDD OF W-ENREG.
     MOVE ZERO TO FLE OF W-ENREG (3), FLE OF W-ENREG (4),
                  FLE OF W-ENREG (5), FLE OF W-ENREG (6).
     IF CODEXIST = 0  MOVE 1 TO FLE  OF W-ENREG (3)
                          MOVE SPACE TO DEB-NU-CPTE OF W-ENREG,
                                        ID-GESTION  OF W-ENREG
        ELSE  PERFORM PROCESS-DEDUCT
              PERFORM PROCESS-CDD-EXISTANT.
     CALL PW1  USING  F2-RECORD, EOF-F2.
     IF EOF-F2 = 1  OR  ID-ENREG OF ENREG-F2 = 0
                  MOVE 1 TO EOF-CONTRAT,EOF-CREANCIER
        ELSE  IF ID-ENREG OF ENREG-F2 = 2
                  MOVE 1 TO EOF-CONTRAT.
     PERFORM  P-OPERATION  UNTIL  EOF-CONTRAT = 1.
*
 PROCESS-CDD-EXISTANT.
*
              IF OPPCDD = 1  MOVE 1 TO FLE OF W-ENREG (5)
                 ELSE NEXT SENTENCE.  MOVE W-DESC-OP-CDD TO DESC-OP.CDD OF W-ENREG
              IF ETATCDD = 1  MOVE 1 TO FLE OF W-ENREG (4)
                 ELSE NEXT SENTENCE.  MOVE W-DATE- REVOCATION TO DATE-REVOCATION OF
                                                                 W- ENREG
              IF IDENTCRE  NOT =  NU-ID-CRE OF W-ENREG
                 MOVE 1 TO FLE  OF W-ENREG (6)
                 MOVE IDENTCRE TO TMC-NU-ID-CRE OF W-ENREG.
*
 P-OPERATION.
     MOVE ZERO TO FLE OF W-ENREG (7).
     IF NUREF  NOT =  NU-REFERENCE OF OPERATION
              MOVE 1 TO FLE OF W-ENREG (7)
              MOVE NUREF TO TMC-NU-REF OF W-ENREG.
     IF TAB OF W-ENREG  =  ALL ZERO
              MOVE CORR W-ENREG TO ENREG-ACC
              MOVE CORR OPERATION TO ENREG-ACC
              WRITE ENREG-ACC
        ELSE  MOVE CORR W-ENREG TO ENREG-NONACC
              MOVE CORR OPERATION TO ENREG-NONACC
              WRITE ENREG-NONACC.
     CALL PW1 USING  F2-RECORD, EOF-F2.
     IF EOF-F2 = 1 OR  ID-ENREG OF ENREG-F2 = 0
              MOVE 1 TO EOF-CONTRAT, EOF-CREANCIER
     ELSE  IF ID-ENREG OF ENREG-F2 = 2
              MOVE 1 TO EOF-CONTRAT.
*
 PROCESS-DEDUCT.
     MOVE CPTE-DEB  TO  DEB-NU-CPTE OF W-ENREG.
     IF D-CPTE-DEB = 001 AND C-CPTE-DEB NOT < 50000
                         AND NOT INTERVALLE
            MOVE '001N ' TO ID-GESTION OF W-ENREG
     ELSE IF D-CPTE-DEB =001 AND C-CPTE-DEB <50000
                         AND INTERVALLE
              MOVE '001F ' TO ID-GESTION OF W-ENREG
          ELSE IF D-CPTE-DEB = 004 AND C-DEUX = 25
                   MOVE '00425' TO ID-GESTION OF W-ENREG
              ELSE IF D-CPTE-DEB = 011 AND C-DEUX = 25
                       MOVE '01125' TO ID-GESTION OF W-ENREG
                   ELSE IF D-CPTE-DEB = 034
                            MOVE '034N ' TO ID-GESTION OF W-ENREG
                       ELSE IF D-CPTE-DEB = 035
                                MOVE '035F ' TO ID-GESTION OF W-ENREG
```

ELSE
MOVE 1-CHTL-LEN TO ID-GESTION OF W-FLREC.

A N N E X E 5

```
type pgrspe.cbl
 IDENTIFICATION DIVISION.
 PROGRAM-ID. PGRSPE.
*

*
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT FICH-IN ASSIGN TO DSK
      RECORDING MODE IS ASCII
      ORGANIZATION IS SEQUENTIAL.
     SELECT FO-ACC  ASSIGN TO DSK
      RECORDING MODE IS ASCII
      ORGANIZATION IS SEQUENTIAL.
     SELECT FO-NONACC ASSIGN TO DSK
      RECORDING MODE IS ASCII
      ORGANIZATION IS SEQUENTIAL.
*
*
 DATA DIVISION.
*
 FILE SECTION.
 FD  FICH-IN; LABEL RECORDS ARE STANDARD
     VALUE OF ID IS  NOM-EXT-FICH-IN.
 01  DEBUT-CEC.
     02  ID-ENREG       PIC 9.
     02  DATE-PIVOT     PIC 9(6).
     02  PROVENANCE     PIC 9(3).
     02  ORIGINE        REDEFINES PROVENANCE PIC 9(3).
 01  DEBUT-CRE.
     02  ID-ENREG       PIC 9.
     02  FILLER         PIC X(12).
     02  DATE-PIVOT     PIC 9(6).
     02  FILLER         PIC X(15).
     02  PROVENANCE     PIC 9(11).
     02  NU-ID-CRE      PIC 9(11).
     02  ORIGINE        REDEFINES NU-ID-CRE PIC 9(11).
     02  NU-CPTE-CRE    PIC 9(12).
 01  OP-CEC.
     02  ID-ENREG       PIC 9.
     02  FILLER         PIC X(21).
     02  NU-DOMI.
        03  DEBUT-NU-DOMI     PIC 9(3).
        03  FIN-NU-DOMI       PIC 9(9).
     02  NU-CPTE-CRE    PIC 9(12).
     02  NATURE-OP      PIC 9.
     02  MONTANT-OP     PIC 9(12).
     02  NOM-CRE        PIC X(26).
     02  NU-ID-CRE      PIC 9(11).
     02  COMM1          PIC X(15).
     02  COMM2          PIC X(15).
     02  NU-REFERENCE PIC 9(12).
 01  OP-CRE.
     02  ID-ENREG       PIC 9.
     02  FILLER         PIC X(6).
     02  NU-DOMI.
        03  DEBUT-NU-DOMI     PIC 9(3).
        03  FIN-NU-DOMI       PIC 9(9).
     02  NATURE-OP      PIC 9.
     02  MONTANT-OP     PIC 9(12).
     02  NOM-CRE        PIC X(26).
     02  COMM1          PIC X(15).
```

```
      02  COMM2          PIC X(15).
      02  NU-REFERENCE PIC 9(12).
  01  ENREG-FICH-IN.
      02 ID-ENREG        PIC 9.
*
 FD FO-ACC; LABEL RECORDS ARE STANDARD
            VALUE OF ID IS NOM-EXT-FO-ACC.
  01  ENREG-ACC.
      02  DATE-PIVOT                          PIC 9(6).
      02  PROVENANCE                          PIC 9(11).
      02  ORIGINE                             PIC 9(11).
      02  NU-ID-CRE                           PIC 9(11).
      02  NU-CPTE-CRE                         PIC 9(12).
      02  NU-DOMI                             PIC 9(12).
      02  NATURE-OP                           PIC 9.
      02  MONTANT-OP                          PIC 9(12).
      02  NOM-CRE                             PIC X(26).
      02  COMM1          PIC X(15).
      02  COMM2          PIC X(15).
      02  NU-REFERENCE                        PIC 9(12).
      02  DEB-NU-CPTE                         PIC 9(12).
      02  ID-GESTION                          PIC X(5).
*
 FD FO-NONACC; LABEL RECORDS ARE STANDARD
               VALUE OF ID IS NOM-EXT-FO-NONACC.
  01  ENREG-NONACC.
      02  DATE-PIVOT                          PIC 9(6).
      02  PROVENANCE                          PIC 9(11).
      02  ORIGINE                             PIC 9(11).
      02  NU-ID-CRE                           PIC 9(11).
      02  NU-CPTE-CRE                         PIC 9(12).
      02  NU-DOMI                             PIC 9(12).
      02  NATURE-OP                           PIC 9.
      02  MONTANT-OP                          PIC 9(12).
      02  NOM-CRE                             PIC X(26).
      02  COMM1          PIC X(15).
      02  COMM2          PIC X(15).
      02  NU-REFERENCE                        PIC 9(12).
      02  DEB-NU-CPTE                         PIC 9(12).
      02  ID-GESTION                          PIC X(5).
      02  TABLEAU                             PIC 9(7).
      02  NU-ID-CRE-CDD                       PIC 9(11).
      02  NU-REF-CDD                          PIC 9(12).
      02  DATE-REVOCATION                     PIC 9(6).
      02  DESC-OPPOSITION-CRE                 PIC X(30).
      02  DESC-OPPOSITION-CDD                 PIC X(30).
*
 WORKING-STORAGE SECTION.
*
 77  NOM-EXT-FICH-IN                          PIC X(9).
 77  NOM-EXT-FO-ACC                           PIC X(9).
 77  NOM-EXT-FO-NONACC                        PIC X(9).
 01  EOF-FICH-IN                              PIC 9.
 01  DCREANCIER.
     02  ETAT-CRE                       PIC 9.
         88  CREANCIER-VERIFIE          VALUE 1.
         88  CREANCIER-NON-VERIFIE      VALUE 0.
     02  NU-ID-CRE                      PIC 9(11).
     02  OPPOSITION-CRE                 PIC 9.
         88  EXISTE-OPPOSITION-CRE      VALUE 1.
         88  PAS-OPPOSITION-CRE         VALUE 0.
     02  EXISTE-CRE                     PIC 9.
         88  CREANCIER-EXISTE           VALUE 1.
         88  CREANCIER-EXISTE-PAS       VALUE 0.
     02  DESC-OPPOSITION-CRE            PIC X(30).
 01  DCONTRAT.
```

```
      02  NU-DOMI                          PIC 9(12).
      02  EXISTE-CDD                       PIC 9.
          88  CDD-EXISTE                   VALUE 1.
          88  CDD-EXISTE-PAS               VALUE 0.
      02  NU-ID-CRE-CDD                    PIC 9(11).
      02  NU-REF-CDD                       PIC 9(12).
      02  OPPOSITION-CDD                   PIC 9.
          88  EXISTE-OPPOSITION-CDD        VALUE 1.
          88  PAS-OPPOSITION-CDD           VALUE 0.
      02  ETAT-CDD                         PIC 9.
          88  CONTRAT-REVOQUE              VALUE 1.
          88  CONTRAT-EN-COURS             VALUE 0.
      02  DATE-REVOCATION                  PIC 9(6).
      02  DESC-OPPOSITION-CDD              PIC X(30).
      02  DEB-NU-CPTE                      PIC 9(12).
      02  CPTE-DEB  REDEFINES  DEB-NU-CPTE.
          03  D-CPTE-DEB                   PIC 9(3).
          03  C-CPTE-DEB.
              05  C-DEUX                   PIC 9(2).
              05  C-CINQ                   PIC 9(5).
              88  INTERVALLE VALUES ARE 0052000 THRU 0052999.
          03  F-CPTE-DEB                   PIC 9(2).
      02  ID-GESTION                       PIC X(5).
  01  DENTETE.
      02  DATE-PIVOT                       PIC 9(6).
      02  PROVENANCE                       PIC 9(11).
      02  ORIGINE                          PIC 9(11).
  01  DOPERATION.
      02  NU-CPTE-CRE                      PIC 9(12).
      02  NATURE-OP                        PIC 9.
      02  MONTANT-OP                       PIC 9(12).
      02  NOM-CRE                          PIC X(26).
      02  COMM1                            PIC X(15).
      02  COMM2                            PIC X(15).
      02  NU-REFERENCE                     PIC 9(12).
  01  W-TABLEAU                              PIC 9(7).
  01  TAB   REDEFINES   W-TABLEAU.
      02  ELE   OCCURS 7 TIMES  PIC 9.
*
*
 PROCEDURE DIVISION.
*
*
 PROCESS-FICH-IN.
      PERFORM LIRE-NOM-FICH.
      OPEN INPUT FICH-IN.
      OPEN OUTPUT FO-ACC, FO-NONACC.
      MOVE  0 TO EOF-FICH-IN.
      READ FICH-IN RECORD AT END MOVE 1 TO EOF-FICH-IN.
      PERFORM P-GROUPE UNTIL EOF-FICH-IN = 1.
      CLOSE FICH-IN, FO-ACC, FO-NONACC.
      STOP RUN.
*
 LIRE-NOM-FICH.
      DISPLAY  'VEUILLEZ DONNER LES NOMS EXTERNES DES FICHIERS'.
      DISPLAY  "MAX 9 CHAR : FFFFFFSSS DEVIENDRA FFFFFF.SSS".
      DISPLAY  "FICHIER FICH-IN : " WITH NO ADVANCING.
      ACCEPT  NOM-EXT-FICH-IN.
      DISPLAY  'FICHIER FO-ACC : " WITH NO ADVANCING.
      ACCEPT  NOM-EXT-FO-ACC.
      DISPLAY  'FICHIER FO-NONACC : " WITH NO ADVANCING.
      ACCEPT  NOM-EXT-FO-NONACC.
*
 P-GROUPE.
      IF PROVENANCE OF DEBUT-CEC = 102  PERFORM  P-GROUPE-CEC
 ELSE  PERFORM  P-GROUPE-CRE.
```

```
*
 P-GROUPE-CEC.
      MOVE CORR DEBUT-CEC TO DENTETE.
      MOVE ORIGINE OF DEBUT-CEC TO ORIGINE OF DENTETE.
      READ FICH-IN RECORD AT END PERFORM JAMAIS-EXECUTE.
      PERFORM P-GROUPE-CREANCIER-CEC UNTIL ID-ENREG OF ENREG-FICH-IN = 9.
      READ FICH-IN RECORD AT END MOVE 1 TO EOF-FICH-IN.
*
 P-GROUPE-CREANCIER-CEC.
      MOVE NU-ID-CRE OF OP-CEC TO NU-ID-CRE OF DCREANCIER.
      MOVE 0 TO ETAT-CRE.
      PERFORM P-GROUPE-CONTRAT-CEC UNTIL  ID-ENREG OF ENREG-FICH-IN = 9
                OR  NU-ID-CRE OF OP-CEC  NOT =  NU-ID-CRE OF DCREANCIER.
*
 P-GROUPE-CONTRAT-CEC.
      IF DEBUT-NU-DOMI OF OP-CEC  =  048
           PERFORM  VERIF-P-GPE-CONT-CEC-D-CGER
      ELSE PERFORM  P-GROUPE-CONTRAT-CEC-DEST-CEC.
*
 VERIF-P-GPE-CONT-CEC-D-CGER.
      IF CREANCIER-NON-VERIFIE
           PERFORM  VERIFICATION-NU-ID-CRE.
      PERFORM  P-GROUPE-CONTRAT-CEC-DEST-CGER.
*
 P-GROUPE-CONTRAT-CEC-DEST-CGER.
      MOVE NU-DOMI OF OP-CEC TO NU-DOMI OF DCONTRAT.
      PERFORM VERIFICATION-NU-DOMI.
      PERFORM P-OPERATION-CEC-DEST-CGER UNTIL ID-ENREG OF ENREG-FICH-IN =
 9
                OR  NU-DOMI OF OP-CEC  NOT =  NU-DOMI OF DCONTRAT.
*
 P-OPERATION-CEC-DEST-CGER.
      PERFORM  CALCUL-DESC-COMPLETEE-CEC.
      IF  TAB  =  ALL ZERO
          MOVE CORR DCREANCIER TO ENREG-ACC
          MOVE CORR DCONTRAT TO ENREG-ACC
          MOVE CORR DENTETE TO ENREG-ACC
          MOVE CORR DOPERATION TO ENREG-ACC
          WRITE ENREG-ACC
      ELSE MOVE CORR DCREANCIER TO ENREG-NONACC
           MOVE CORR DCONTRAT TO ENREG-NONACC
           MOVE CORR DENTETE TO ENREG-NONACC
           MOVE CORR DOPERATION TO ENREG-NONACC
           MOVE W-TABLEAU TO TABLEAU OF ENREG-NONACC
           WRITE ENREG-NONACC.
      READ FICH-IN RECORD AT END PERFORM JAMAIS-EXECUTE.
*
 P-GROUPE-CONTRAT-CEC-DEST-CEC.
      MOVE NU-DOMI OF OP-CEC TO NU-DOMI OF DCONTRAT.
      PERFORM P-OPERATION-CEC-DEST-CEC UNTIL ID-ENREG OF ENREG-FICH-IN =
 9
                OR  NU-DOMI OF OP-CEC  NOT =  NU-DOMI OF DCONTRAT.
*
 P-OPERATION-CEC-DEST-CEC.
      READ FICH-IN RECORD AT END PERFORM JAMAIS-EXECUTE.
*
*-----------------------------------------------------------------------
*
 P-GROUPE-CRE.
      MOVE CORR DEBUT-CRE TO DENTETE.
      MOVE NU-ID-CRE   OF DEBUT-CRE TO NU-ID-CRE   OF DCREANCIER.
      MOVE NU-CPTE-CRE OF DEBUT-CRE TO NU-CPTE-CRE OF DOPERATION.
      MOVE ORIGINE OF DEBUT-CRE TO ORIGINE OF DENTETE.
      READ FICH-IN RECORD AT END PERFORM JAMAIS-EXECUTE.
      MOVE 0 TO ETAT-CRE.
```

```
     PERFORM P-GROUPE-CONTRAT-CRE UNTIL  ID-ENREG OF ENREG-FICH-IN = 9.
     READ FICH-IN RECORD AT END MOVE 1 TO EOF-FICH-IN.
*
 P-GROUPE-CONTRAT-CRE.
     IF DEBUT-NU-DOMI OF OP-CRE  =  048
          PERFORM  VERIF-P-GPE-CONT-CRE-D-CGER
     ELSE PERFORM  P-GROUPE-CONTRAT-CRE-DEST-CEC.
*
 VERIF-P-GPE-CONT-CRE-D-CGER.
     IF CREANCIER-NON-VERIFIE
          PERFORM  VERIFICATION-NU-ID-CRE.
     PERFORM  P-GROUPE-CONTRAT-CRE-DEST-CGER.
*
 P-GROUPE-CONTRAT-CRE-DEST-CGER.
     MOVE NU-DOMI OF OP-CRE TO NU-DOMI OF DCONTRAT.
     PERFORM VERIFICATION-NU-DOMI.
     PERFORM P-OPERATION-CRE-DEST-CGER UNTIL ID-ENREG OF ENREG-FICH-IN =
 9
              OR  NU-DOMI OF OP-CRE  NOT =  NU-DOMI OF DCONTRAT.
*
 P-OPERATION-CRE-DEST-CGER.
     PERFORM  CALCUL-DESC-COMPLETEE-CRE.
     IF  TAD  =  ALL ZERO
         MOVE CORR DCREANCIER TO ENREG-ACC
         MOVE CORR DCONTRAT TO ENREG-ACC
         MOVE CORR DENTETE TO ENREG-ACC
         MOVE CORR DOPERATION TO ENREG-ACC
         WRITE ENREG-ACC
     ELSE MOVE CORR DCREANCIER TO ENREG-NONACC
          MOVE CORR DCONTRAT TO ENREG-NONACC
          MOVE CORR DENTETE TO ENREG-NONACC
          MOVE CORR DOPERATION TO ENREG-NONACC
          MOVE W-TABLEAU TO TABLEAU OF ENREG-NONACC
          WRITE ENREG-NONACC.
     READ FICH-IN RECORD AT END PERFORM JAMAIS-EXECUTE.
*
 P-GROUPE-CONTRAT-CRE-DEST-CEC.
     MOVE NU-DOMI OF OP-CRE TO NU-DOMI OF DCONTRAT.
     PERFORM P-OPERATION-CRE-DEST-CEC UNTIL ID-ENREG OF ENREG-FICH-IN =
 9
              OR  NU-DOMI OF OP-CRE  NOT =  NU-DOMI OF DCONTRAT.
*
 P-OPERATION-CRE-DEST-CEC.
     READ FICH-IN RECORD AT END PERFORM JAMAIS-EXECUTE.
*
*
 VERIFICATION-NU-ID-CRE.
     CALL CREEXI USING  DCREANCIER.
*
 VERIFICATION-NU-DOMI.
     CALL CDDEXI USING  DCONTRAT.
     IF   CDD-EXISTE-PAS
          MOVE SPACES  TO  DCD-NU-CPTE OF DCONTRAT
          MOVE SPACES  TO  ID-GESTION  OF DCONTRAT
     ELSE
          PERFORM PROCESS-DEDUCT.
*
 CALCUL-DESC-COMPLETEE-CEC.
     MOVE CORR OP-CEC TO DOPERATION.
     MOVE ALL ZERO TO TAB.
     IF CREANCIER-EXISTE-PAS
            MOVE 1 TO ELE OF TAB (1)
     ELSE   IF EXISTE-OPPOSITION-CRE
            MOVE 1 TO ELE OF TAB (2).
     IF CDD-EXISTE-PAS
            MOVE 1 TO ELE OF TAB (3)
```

```
      ELSE    PERFORM FIN-TRT-TAB.
*
*
 CALCUL-DESC-COMPLETEE-CRE.
      MOVE CORR OP-CRE TO DOPERATION.
      MOVE ALL ZERO TO TAB.
      IF CREANCIER-EXISTE-PAS
             MOVE 1 TO ELE OF TAB (1)
      ELSE   IF EXISTE-OPPOSITION-CRE
             MOVE 1 TO ELE OF TAB (2).
      IF CDD-EXISTE-PAS
             MOVE 1 TO ELE OF TAB (3)
      ELSE   PERFORM FIN-TRT-TAB.
*
 FIN-TRT-TAB.
      IF CONTRAT-REVOQUE                    MOVE 1 TO ELE OF TAB (4).
      IF EXISTE-OPPOSITION-CDD              MOVE 1 TO ELE OF TAB (5).
      IF NU-ID-CRE OF DCREANCIER NOT =      NU-ID-CRE-CDD OF DCONTRAT
                                            MOVE 1 TO ELE OF TAB (6).
      IF NU-REFERENCE OF DOPERATION  NOT =  NU-REF-CDD OF DCONTRAT
                                            MOVE 1 TO ELE OF TAB (7).
*
 PROCESS-DEDUCT.
      IF D-CPTE-DEB = 001 AND C-CPTE-DEB NOT < 50000
                          AND NOT INTERVALLE
             MOVE '001N ' TO ID-GESTION OF DCONTRAT
      ELSE IF D-CPTE-DEB =001 AND C-CPTE-DEB <50000
                              AND INTERVALLE
                MOVE '001F ' TO ID-GESTION OF DCONTRAT
           ELSE IF D-CPTE-DEB = 004 AND C-DEUX = 25
                     MOVE '00425' TO ID-GESTION OF DCONTRAT
                ELSE IF D-CPTE-DEB = 011 AND C-DEUX = 25
                          MOVE '01125' TO ID-GESTION OF DCONTRAT
                     ELSE IF D-CPTE-DEB = 034
                               MOVE '034N ' TO ID-GESTION OF DCONTRAT
                          ELSE IF D-CPTE-DEB = 035
                                    MOVE '035F ' TO ID-GESTION OF DCONTR
AT
                               ELSE
                               MOVE D-CPTE-DEB TO ID-GESTION OF DCONTRAT
.
*
*
 JAMAIS-EXECUTE.
*
* Ce paragraphe a ete cree parce que la syntaxe COBOL du read impose
* l'existence d'une instruction imperative AT END qui sera executee
* lorsqu'on arrive en fin de fichier.
* Si le fichier d'entree est conforme a la structure donnee a la page ?
* l'effet du READ a cet endroit du programme ne provoquera jamais
* l'execution de l'instruction imperative puisque conformement a la
* structure du fichier d'entree on ne devrait jamais etre en fin de
* fichier a cet endroit.
  EXIT.
@
```

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CREEXI.
*
*
 DATA DIVISION.
*
 LINKAGE SECTION.
 01  DCREANCIER.
     02  ETAT-CREANCIER                    PIC 9.
     02  IDENT-CREANCIER                   PIC 9(11).
     02  OPPOSITION-CREANCIER              PIC 9.
     02  CREANCIER-EXISTE                  PIC 9.
     02  DESC-OP-CREANCIER                 PIC X(30).
*
*
 PROCEDURE DIVISION.
*
 ENTRY 'CREEXI' USING DCREANCIER.

*
 MOVE 1 TO ETAT-CREANCIER.
 DISPLAY "Pour ce numero d identification de creancier : " WITH NO ADVANCING.
 DISPLAY IDENT-CREANCIER.
 DISPLAY "Taper 1 si il existe,0 sinon suivi de <return>".
 ACCEPT  CREANCIER-EXISTE.
 IF  CREANCIER-EXISTE = 1
     DISPLAY "Taper 1 si il y a une opposition sur ce creancier "
     DISPLAY " 0 sinon suivi de <return>"
     ACCEPT  OPPOSITION-CREANCIER
     IF OPPOSITION-CREANCIER = 1
        DISPLAY "Donnez la description de l opposition"
        ACCEPT DESC-OP-CREANCIER.
 EXIT PROGRAM.
```

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CDDEXI.
*
*
 DATA DIVISION.
*
 LINKAGE SECTION.
*
 01  DCONTRAT.
     02  NU-DOMI                    PIC 9(12).
     02  CDDEXIST                   PIC 9.
     02  IDENTCRE                   PIC 9(11).
     02  NUREF                      PIC 9(12).
     02  OPPCDD                     PIC 9.
     02  ETATCDD                    PIC 9.
     02  DATE-REVOCATION            PIC 9(6).
     02  DESC-OP-CDD                PIC X(30).
     02  DEB-NU-CPTE                PIC 9(12).
     02  ID-GESTION                 PIC X(5).
*
*
 PROCEDURE DIVISION.
*
 ENTRY  'CDDEXI'  USING DCONTRAT.

*
 DISPLAY "Pour ce numero de domiciliation : " WITH NO ADVANCING.
 DISPLAY  NU-DOMI.
 DISPLAY "Taper 1 si il existe,0 sinon suivi de <return> :".
 ACCEPT  CDDEXIST.
 IF  CDDEXIST  = 1
     DISPLAY "Donnez le numero d identification du creancier : "
     ACCEPT  IDENTCRE
     DISPLAY "Donnez le numero de reference : "
     ACCEPT  NUREF
     DISPLAY "Taper 1 si il y a une opposition sur ce contrat,0 sinon : "
     ACCEPT  OPPCDD
     PERFORM TRT-OPPOSITION
     DISPLAY "Taper 0 si l etat du contrat est en cours, 1 si revoque : "
     ACCEPT  ETATCDD
     PERFORM TRT-ETATCDD
     DISPLAY "Donnez le numero de cpte debiteur se trouvant sur ce contrat : '
     ACCEPT  DEB-NU-CPTE.
 EXIT PROGRAM.
*
 TRT-OPPOSITION.
     IF OPPCDD = 1
        DISPLAY "Donnez la description de l opposition"
        ACCEPT DESC-OP-CDD.
*
 TRT-ETATCDD.
     IF ETATCDD = 1
        DISPLAY "Donner la date de revocation"
        ACCEPT DATE-REVOCATION.
```