Le Projet M.A.C.S.I.-P : Méthode d'Aide à la Conception des Systèmes d'Informations orientée vers la fabrication de Prototypes d'applications informatiques de gestion

Jean-Pierre Giraudin

▶ **To cite this version:**

Jean-Pierre Giraudin. Le Projet M.A.C.S.I.-P : Méthode d'Aide à la Conception des Systèmes d'Informations orientée vers la fabrication de Prototypes d'applications informatiques de gestion. Modélisation et simulation. Université Joseph-Fourier - Grenoble I, 1977. Français. NNT : . tel-00287557

HAL Id: tel-00287557

https://theses.hal.science/tel-00287557

Submitted on 12 Jun 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE

présentée à

Université Scientifique et Médicale de Grenoble

pour obtenir le grade de
DOCTEUR DE 3ème CYCLE
«Informatique»

par

Jean-Pierre GIRAUDIN

LE PROJET M.A.C.S.I.-P.

METHODE D'AIDE A LA CONCEPTION DES SYSTEMES D'INFORMATIONS ORIENTEE VERS LA FABRICATION DE PROTOTYPES D'APPLICATIONS INFORMATIQUES DE GESTION.

Thèse soutenue le 1 juillet 1977 devant la Commission d'Examen :

Président : C. DELOBEL

Rapporteur : F. PECCOUD

Examinateur : J. KOULOUMDJIAN

UNIVERSITE SCIENTIFIQUE
ET MEDICALE DE GRENOBLE

Monsieur Gabriel CAU : Président
Monsieur Pierre JULLIEN : Vice Président

MEMBRES DU CORPS ENSEIGNant DE L'U.S.M.G.

PROFESSEURS TITULAIRES

MM	AMBLARD Pierre	Clinique de dermatologie
	ARNAUD Paul	Chimie
	ARVIEU Robert	I.S.N
	AUBERT Guy	Physique
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique Expérimentale
	BARBIER Reynold	Géologie appliquée
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale
	BEAUDOING André	Clinique de Pédiatrie et Puériculture
	BELORIZKY Elie	Physique
	BERNARD Alain	Mathématiques Pures
Mme	BERTRANDIAS Françoise	Mathématiques Pures
MM.	BERTRANDIAS Jean-Paul	Mathématiques Pures
	BEZEZ Henri	Pathologie chirurgicale
	BLAMBERT Maurice	Mathématiques Pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET-EYMARD Joseph	Clinique gastro-entérologique
Mme	BONNIER Marie-Jeanne	Chimie générale
MM.	BOUCHERLE André	Chimie et toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BOUSSARD Jean-Claude	Mathématiques appliquées
	BOUTET DE MONTVEL Louis	Mathématiques Pures
	BRAVARD Yves	Géographie
	CABANEL Guy	Clinique rhumatologique et hydrologique
	CALAS François	Anatomie
	CARLIER Georges	Biologie végétale
	CARRAZ Gilbert	Biologie animale et pharmacodynamie
	CAU Gabriel	Médecine légale et toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques Pures
	CHARACHON Robert	Clinique Oto-rhino-laryngologique
	CHATEAU Robert	Clinique de neurologie
	CHIBON Pierre	Biologie animale
	COEUR André	Pharmacie chimique et chimie analytique
	CONTAMTIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie pathologique
Mme	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBELMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DELORMAS Pierre	Pneumophtisiologie

MM.	DEPORTES Charles	Chimie minérale
	DESRE Pierre	Métallurgie
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée (IUT I)
	DOLIQUE Jean-Michel	Physique des plasmas
	DREYFUS Bernard	Thermodynamique
	DUCROS Pierre	Cristallographie
	GAGNAIRE Didier	Chimie Physique
	GALVANI Octave	Mathématiques Pures
	GASTINEL Noël	Analyse numérique
	GAVEND Michel	Pharmacologie
	GEINDRE Michel	Electroradiologie
	GERBER Robert	Mathématiques Pures
	GERMAIN Jean-Pierre	Mécanique
	GIRAUD Pierre	Géologie
	JANIN Bernard	Géographie
	KAHANE André	Physique générale
	KLEIN Joseph	Mathématiques Pures
	KOSZUL Jean-Louis	Mathématiques Pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques Appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
Mme	LAJZEROWICZ Janine	Physique
MM.	LAJZEROWICZ Joseph	Physique
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie Pharmaceutique
	LAURENT Pierre	Mathématiques Appliquées
	LEDRU Jean	Clinique médicale B
	LE ROY Philippe	Mécanique (IUT I)
	LLIBOUTRY Louis	Géophysique
	LOISEAUX Pierre	Sciences Nucléaires
	LONGEQUEUE Jean-Pierre	Physique Nucléaire
	LOUP Jean	Géographie
Melle	LUTZ Elisabeth	Mathématiques Pures
MM.	MALINAS Yves	Clinique Obstétricale
	MARTIN-NOEL Pierre	Clinique Cardiologique
	MAZARE Yves	Clinique Médicale A
	MICHEL Robert	Minéralogie et Pétrographie
	MICOUD Max	Clinique Maladies infectieuses
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie Nucléaire
	NOZIERES Philippe	Spectrometrie Physique
	OZENDA Paul	Botanique
	PAYAN Jean-Jacques	Mathématiques Pures
	PEBAY-PEYROULA Jean-Claude	Physique
	PERRET Jean	Semeiologie Médicale (Neurologie)
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	REVOL Michel	Urologie
	RINALDI Renaud	Physique
	DE ROUGEMONT Jacques	Neuro-Chirurgie
	SEIGNEURIN Raymond	Microbiologie et Hygiène
	SENGEL Philippe	Zoologie
	SIBILLE Robert	Construction mécanique (IUT I)
	SOUTIF Michel	Physique générale
	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale

MM.	VAILLANT François	Zoologie
	VALENTIN Jacques	Physique Nucléaire
	VAUQUOIS Bernard	Calcul électronique
Mme	VERAIN Alice	Pharmacie galénique
MM.	VERAIN André	Physique
	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale

PROFESSEURS ASSOCIES

MM.	CRABBE Pierre	CERMO
	DEMBICKI Eugéniuz	Mécanique
	JOHNSON Thomas	Mathématiques appliquées
	PENNEY Thomas	Physique

PROFESSEURS SANS CHAIRE

Mle	AGNIUS-DELORD Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	AMBROISE-THOMAS Pierre	Parasitologie
	ARMAND Gilbert	Géographie
	BENZAKEN Claude	Mathématiques appliquées
	BIAREZ Jean-Pierre	Mécanique
	BILLET Jean	Géographie
	BOUCHET Yves	Anatomie
	BRUGEL Lucien	Energétique (IUT I)
	BUISSON René	Physique (IUT I)
	BUTEL Jean	Orthopédie
	COHEN ADDAD Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie
	CONTE René	Physique (IUT I)
	DELOBEL Claude	M.I.A.G.
	DEPASSEL Roger	Mécanique des fluides
	FONTAINE Jean-Marc	Mathématiques Pures
	GAUTRON René	Chimie
	GIDON Paul	Géologie et Minéralogie
	GLENAT René	Chimie organique
	GROULADE Joseph	Biochimie médicale
	HACQUES Gérard	Calcul numérique
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Hygiène et Médecine préventive
	IDELMAN Simon	Physiologie animale
	JOLY Jean-René	Mathématiques Pures
	JULLIEN Pierre	Mathématiques Appliquées
Mme	KAHANE Josette	Physique
MM.	KRAKOWIACK Sacha	Mathématiques Appliquées
	KUHN Gérard	Physique (IUT I)
	LUU DUC Cuong	Chimie organique
	MAYNARD Roger	Physique du solide
Mme	MINIER Colette	Physique (IUT I)
MM.	PELMONT Jean	Biochimie
	PERRIAUX Jean-Jacques	Géologie et Minéralogie
	PFISTER Jean-Claude	Physique du solide
Mle	PIERY Yvette	Physiologie animale

MM.	RAYNAUD Hervé	M.I.A.G.
	REBECQ Jacques	Biologie (CUS)
	REYMOND Jean-Charles	Chirurgie générale
	RICHARD Lucien	Biologie végétale
Mme	RINAUDO Marguerite	Chimie macromoléculaire
MM.	ROBERT André	Chimie papetière
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIROT Louis	Chirurgie générale
Mme	SOUTIF Jeanne	Physique générale
MM.	STIEGLITZ Paul	Anesthésiologie
	VIALON Pierre	Géologie
	VAN CUTSEM Bernard	Mathématiques Appliquées

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

MM.	ARMAND Yves	Chimie (IUT I)
	BACHELOT Yvan	Endocrinologie
	BARGE Michel	Neuro chirurgie
	BEGUIN Claude	Chimie organique
Mme	BERIEL Hélène	Pharmacodynamie
MM.	BOST Michel	Pédiatrie
	BOUCHARLAT Jacques	Psychiatrie adultes
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BRODEAU François	Mathématiques (IUT B) (Personne étrangère habilitée à être directeur de thèse)
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse
	CHARDON Michel	Géographie
	CHERADAME Hervé	Chimie papetière
	CHIAVERINA Jean	Biologie appliquée (EFP)
	CONTAMIN Charles	Chirurgie thoracique et cardio-vasculaire
	CORDONNIER Daniel	Néphrologie
	COULOMB Max	Radiologie
	CROUZET Guy	Radiologie
	CYROT Michel	Physique du solide
	DENIS Bernard	Cardiologie
	DOUCE Roland	Physiologie végétale
	DUSSAUD René	Mathématiques (CUS)
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	FAURE Gilbert	Urologie
	GAUTIER Robert	Chirurgie générale
	GIDON Maurice	Géologie
	GROS Yves	Physique (IUT I)
	GUIGNIER Michel	Thérapeutique
	GUITTON Jacques	Chimie
	HICTER Pierre	Chimie
	JALBERT Pierre	Histologie
	JUNIEN-LAVILLAVROY Claude	O.R.L.
	KOLODIE Lucien	Hématologie
	LE NOC Pierre	Bactériologie-virologie
	MACHE Régis	Physiologie végétale
	MAGNIN Robert	Hygiène et médecine préventive
	MALLION Jean-Michel	Médecine du travail

MM.	MARECHAL Jean	Mécanique (IUT I)
	MARTIN-BOUYER Michel	Chimie (CUS)
	MICHOULIER Jean	Physique (IUT I)
	NEGRE Robert	Mécanique (IUT I)
	NEMOZ Alain	Thermodynamique
	NOUGARET Marcel	Automatique (IUT I)
	PARAMELLE Bernard	Pneumologie
	PECCOUD François	Analyse (IUT B) (Personnalité étrangère habilitée à être directeur de thèse)
	PEFFEN René	Métallurgie (IUT I)
	PERRIER Guy	Géophysique-Glaciologie
	PHELIP Xavier	Rhumatologie
	RACHAIL Michel	Médecine Interne
	RACINET Claude	Gynécologie et Obstétrique
	RAMBAUD André	Hygiène et Hydrologie (Pharmacie)
	RAMBAUD Pierre	Pédiatrie
	RAPHAEL Bernard	Stomatologie
Mme	RENAUDET JAcqueline	Bactériologie (Pharmacie)
MM	ROBERT Jean-Bernard	Chimie Physique
	ROMIER Guy	Mathématiques (IUT B) (Personnalité étrangère habilité à être directeur de thèse)
	SCHAERER René	Cancérologie
	SHOM Jean-Claude	Chimie Générale
	STOEBNER Pierre	Anatomie Pathologie
	VROUSOS Constantin	Radiologie

MAITRESDE CONFERENCESASSOCIES

MM.	DEVINE Roderick	Spectro Physique
	HODGES Christopher	Transition de Phases

Fait à SAINT MARTIN D'HERES, NOVEMBRE 1976

Je tiens à remercier :

Monsieur le Professeur Claude DELOBEL pour l'intérêt qu'il a porté à ce travail et pour l'honneur qu'il m'a fait de présider ce jury;

Monsieur Jacques KOULOUMDJIAN, Maître de Conférences à l'Université Claude Bernard de Lyon, qui a accepté de juger cette thèse et dont les critiques bienveillantes m'ont été précieuses;

Monsieur François PECCOUD, Maître de Conférences à l'Université des Sciences Sociales de Grenoble, qui est à l'origine de ce projet. Je veux aussi lui exprimer ma plus profonde gratitude pour son accueil dans l'équipe lors de conditions difficiles, pour la confiance qu'il m'a toujours témoignée et pour ses nombreux conseils et encouragements sans lesquels ce travail n'aurait pas été possible.

Je suis également reconnaissant :

à Michel ADIBA, Alain BONJEAN, Xavier CASTELLANI, Jean-Claude FAVRE, Dominique JAHU, Claude PAOLI, ainsi qu'à mes collègues de l'I.U.T., comme à tous les ingénieurs de l'Institut de Formation et de Conseil, pour leur compétence et leur collaboration apportées au niveau de la réalisation technique de ce travail;

à ceux qui, par leur travail d'animation du groupe INFORSID, m'ont permis de participer à de nombreuses discussions avec des chercheurs et des utilisateurs sur des sujets proches de mes préoccupations;

à Marie José DOREL qui a assuré la dactylographie de ce texte avec soin, rapidité et sympathie, avec l'aide de Fabienne BOULIER;

au Service de Reprographie pour la réalisation matérielle de cet ouvrage.

Je ne saurais oublier ,

dans cette traditionnelle "page de remerciements" , Agnès, ma femme, pour son soutien moral constant et son aide indispensable lors des corrections de cette thèse.

A mes parents.

"Par méthode, j'entends des règles certaines et faciles, grâce auxquelles tous ceux qui les observent exactement ne supposeront jamais vrai ce qui est faux et parviendront sans se fatiguer en efforts inutiles, mais en accroissant progressivement leur science, à la connaissance vraie de ce qu'ils peuvent atteindre".

DESCARTES
Règles pour la direction de l'esprit

SOMMAIRE

Chapitre III - UN LANGAGE DE DESCRIPTION

Chapitre IV - UN OUTIL : LE LOGICIEL MACSI-P

Annexe B - Le projet MACSI-1

Annexe C - Réalisations techniques

CHAPITRE I

INTRODUCTION

MACSI-P est une Méthode d'Aide à la Conception des Systèmes d'Informations orientée vers la fabrication de Prototypes d'applications informatiques de gestion.

I-1 - EVOLUTION DES TECHNIQUES

Il faut constater que jusqu'à présent, de nombreuses applications informatiques de gestion mises en place n'utilisent qu'une technologie très élémentaire essentiellement caractérisée par :

- des saisies sur cartes perforées, ou bandes magnétiques, ou disquettes ;
- des traitements sur fichiers en séquentiel, ou séquentiel indexé, ou direct, impliquant naturellement des données peu complexes, des algorithmes relativement simples et des langages de programmation avec lesquels toute la cinématique d'accès aux données est à programmer.

Les nombreuses méthodes d'analyse commercialisées [L8], orientées vers un mode d'exploitation à traitement différé, répondent assez bien à ces critères techniques.

Aujourd'hui, nous constatons une évolution de l'informatique qui fait apparaître un nombre de plus en plus important de ressources techniques. Citons les principales :

- au niveau des matériels :
  - . la construction d'unités centrales du type microprocesseur ;
  - . le remplacement des mémoires centrales à tores par des mémoires à circuits intégrés (ROM, RAM, PROM, ...) ;

- l'introduction de nouvelles mémoires secondaires à bandes magnétiques (cassettes), à disques (disquettes) et demain à bulles ;
- le développement des transmissions par téléphone à des débits très variés dans des réseaux internationaux (CYCLADES, TRANSPAC, EURONET, ...) ;
- l'évolution des organes d'entrée et de sortie avec la commercialisation de périphériques programmables manipulant des données sur une grande diversité de supports (papier, magnétique, écran cathodique, microfilm, ...).

- au niveau des logiciels :
  - le choix de logiciels de base programmés ou microprogrammés ;
  - la multiplication des langages de programmation et des langages spécialisés ;
  - l'apparition des systèmes de bases de données ;

Devant une telle diversité des moyens techniques, le processus de conception d'une application informatique se trouve alors profondément transformé.

En effet, jusqu'à une date récente, le petit nombre de ressources disponibles pour réaliser une application conduisait les analystes à penser l'application en fonction des possibilités de sa réalisation technique . Ils donnaient donc, très tôt dans la phase de conception, une très grande importance aux moyens techniques à mettre en oeuvre et délaissaient quelque peu l'étude précise des finalités et des conditions d'utilisation de l'application. Les applications étudiées étaient simples dans la mesure où les possibilités de réalisation technique étaient rustiques.

On assiste maintenant à un développement continu des éléments technologiques pour automatiser une fonction de gestion, à des coûts unitaires constamment en baisse. Cette évolution permet d'élargir considérablement le champ des applications potentielles

de l'informatique et leur complexité. Cette situation oblige aussi les analystes d'une application à procéder comme de véritables architectes : ils doivent d'abord répondre au *"POURQUOI"* de l'application pour ne se préoccuper qu'ensuite de son *"COMMENT"*.

On ne peut plus choisir à priori les ressources informatiques mises en oeuvre dans une application sans connaître très précisément d'abord les fonctions qui doivent être remplies par cette application, les services qu'elle doit rendre, les conditions de son utilisation. Quand ces points sont précisés, on peut alors définir les moyens les plus appropriés.

Autrement dit, l'ANALYSE FONCTIONNELLE (définition des fonctions) d'une application doit précéder son ANALYSE ORGANIQUE (choix des organes technologiques à mettre en oeuvre) et s'en distinguer très nettement [L5], [L16], [L22] et [A11].

Notons que de nombreux concepts disparaissent complètement avec un changement de technologie et que d'autres, utilisés sous des vocables distincts dans diverses techniques ont une existence dépassant largement toute technologie particulière. Citons quelques exemples :

- La notion d'"enregistrement" utilisée dans une technique fichier, disparaît avec les systèmes de gestion de bases de données.
- Nous parlons de "document d'entrée perforé" ou d'"édition de listings" avec un traitement différé par lots, et de "protocole de conversation" ou d'"affichage non permanent" sur écran dans le cas d'un traitement temps réel à partir d'un terminal écran.
  Ces notions appartenant à des technologies différentes suggèrent pourtant des propriétés descriptives de données communes pour l'utilisateur.
- Entre des fichiers classiques et des bases de données, il y a bien, malgré des moyens techniques distincts, des problèmes communs de description, de stockage, de mode d'accès et de mise à jour des données.

Notre premier objectif est d'analyser quelles sont les caractéristiques d'une application informatique qui sont indépendantes de telle ou telle technologie particulière et de proposer pour chacune d'elles une description précise.

I-2 - DIFFICULTES DE DEVELOPPEMENT D'UNE APPLICATION INFORMATIQUE DE GESTION

Malgré le développement continu des méthodes d'analyse [A10], [L8], [L16], la conception et la réalisation d'une application informatique sont toujours des opérations difficiles. Citons les principales difficultés qui, à notre idée, conduisent à certains échecs :

- les choix organiques parmi la diversité des technologies existantes dépendent de critères incompatibles (facilités d'emploi, temps de réponse, coûts de mise en oeuvre, ...) ;
- les études de l'organisation environnante et des prévisions d'activité des différents postes de travail sont très difficiles par leur manque d'intégration avec les aspects informatiques de l'application ;
- l'absence de moyens pour assurer la formation et la motivation des utilisateurs (les premiers résultats informatiques effectifs parviennent de nombreux mois après les promesses séduisantes des informaticiens) ;
- la lenteur de la conception et de la réalisation d'une application conduit à des anachronismes inacceptables. Lorsque l'application informatique est enfin définie et réalisée, l'entreprise a évolué et des distorsions apparaissent déjà entre ce qui est réalisé et le nouvel existant.

Pour résoudre ces problèmes, il ne suffit pas de discuter avec les responsables et les utilisateurs potentiels d'une application, ni de choisir l'une des méthodes d'analyse proposées actuellement : seule une REALISATION TECHNIQUE PROVISOIRE permet une maîtrise d'oeuvre d'un projet d'application informatique [L22] [L14] [L15] [A5] [A6] [A14].

C'est notre second objectif que de proposer des moyens logiciels pour réaliser des PROTOTYPES D'APPLICATIONS INFORMATIQUES de GESTION.

Précisons ce que nous entendons par PROTOTYPE.

I-3 - LA METHODE DU PROTOTYPE

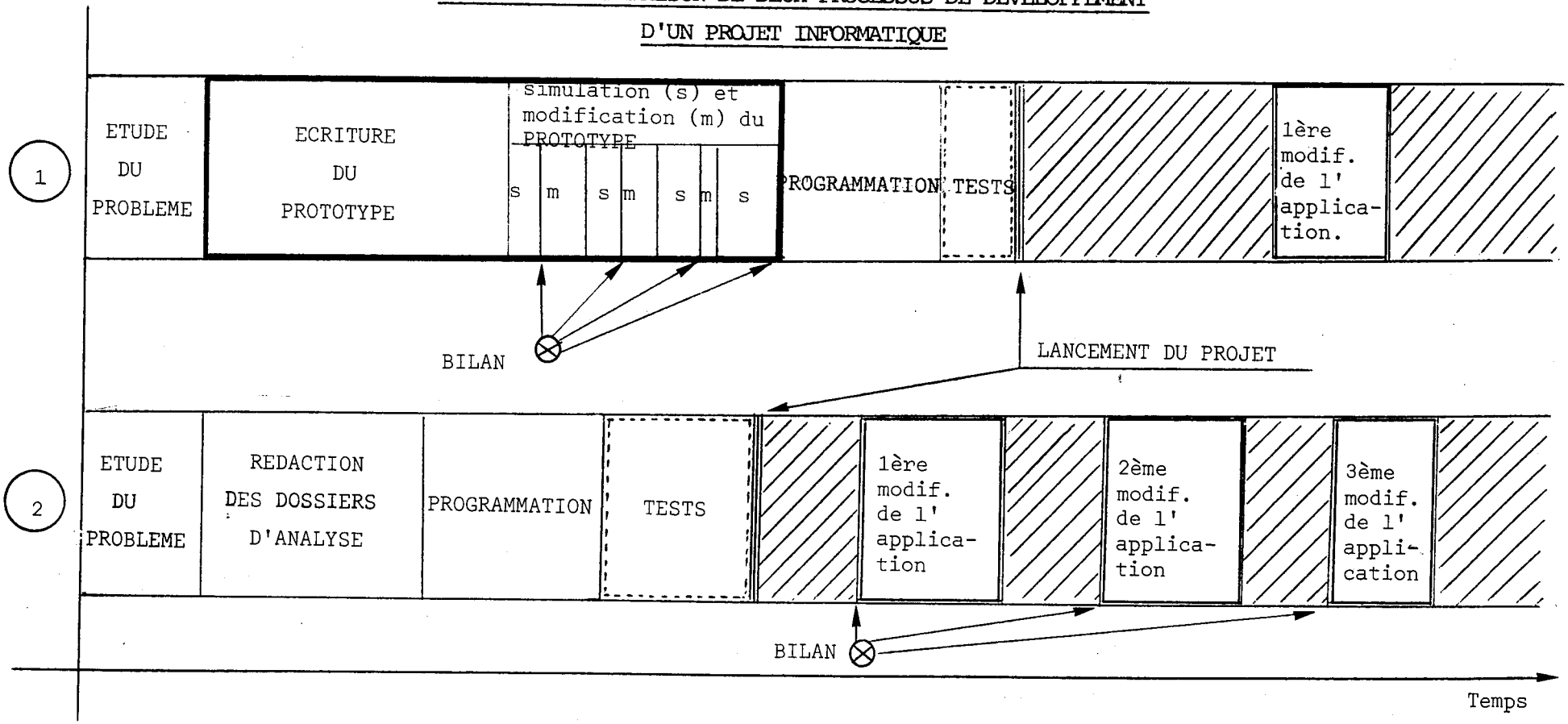
I.3.1 - Les nouvelles étapes de réalisation qu'elle implique

Dans le développement normal d'un projet informatique, on s'accorde généralement pour distinguer plusieurs phases [L8] [L21] [L5] [A24] : étude d'opportunité, analyse fonctionnelle, analyse organique, réalisation, lancement, exploitation et maintenance. A la fin de la phase d'analyse fonctionnelle, il doit y avoir théoriquement un accord précis entre utilisateurs et techniciens informatiques sur les spécifications de l'application. Mais, en fait, on constate souvent que c'est seulement au moment du lancement du projet que peut être réellement évaluée son efficacité : des modifications dans la conception vont être alors exigées rapidement par les services utilisateurs à la lumière des premiers mois d'utilisation de cette application.

Si ces modifications sont trop considérables, le coût de leur réalisation peut représenter une part importante du coût initial d'étude et de développement du projet. Elles seront surtout considérées par les services qui les demandent comme une preuve d'échec de la conception initiale du projet.

C'est, dans ce genre de situation, que la réalisation d'un prototype de l'application a pour enjeu fondamental de permettre **un nouveau processus de développement dans lequel puissent s'exprimer, avant la réalisation de la version opérationnelle, toutes les critiques qu'appelle sa conception.** L'utilisation intensive d'un prototype provoque naturellement des critiques : sa mise au point se poursuit, par approximations successives, jusqu'à ce que les services qui le testent puissent donner leur accord. La réalisation de la version opérationnelle peut alors intervenir. Après son lancement, il est possible d'espérer que l'implication préalable des services avec le prototype et les nombreuses modifications qu'ils lui ont apportées les conduiront à ne pas demander des modifications majeures par la suite (voir figure).

SCHEMA DE COMPARAISON DE DEUX PROCESSUS DE DEVELOPPEMENT D'UN PROJET INFORMATIQUE



Légende :

Exploitation opérationnelle

1 Processus de développement avec réalisation d'un prototype

2 Processus normal de développement d'un projet.

I.3.2 - Prototype d'une application informatique : Définition

Il faut entendre, par **prototype, un modèle de la structure fonctionnelle de l'application.** Nous utiliserons le terme de MODELE au sens de MINSKY :

"Pour un opérateur O, un objet M est un modèle d'un objet A dans la mesure où O peut utiliser M pour répondre aux questions qui l'intéressent au sujet de A".

Ainsi, dans le cadre particulier d'une application informatique, nous pouvons dire :

Pour le futur utilisateur d'une application informatique de gestion, un prototype de cette application est un modèle, dans la mesure où l'utilisateur peut se servir du prototype pour répondre aux questions qui l'intéressent au sujet de la future application.

Essayons donc de faire l'inventaire des principales questions posées par une personne qui participe à la conception d'un projet informatique sachant que, plus tard, elle en sera l'un des principaux utilisateurs.

- Quels sont les avantages de ce projet en termes de services rendus ?
- Quelles données devront être fournies par les services pour que l'application fonctionne ?
- Quelle charge de travail supplémentaire cela donnera-t-il aux services ?
- Ceux-ci ont-ils une compétence suffisante pour assurer les nouvelles procédures administratives ?
- Le coût prévu de développement de l'application justifie-t-il les avantages nouveaux procurés ?
- etc...

Le prototype doit apporter des éléments de réponse à ces questions. Nous pouvons alors considérer que **le prototype est une application informatique qui doit être fonctionnellement équivalente à celle étudiée** dans le projet. Equivalence fonctionnelle signifie que toutes les données en entrée et sortie, ainsi que leur logique de

traitement, doivent être les mêmes entre le prototype et la version finale. Par contre, les moyens mis en oeuvre pour les réaliser diffèrent. Ainsi :

- le choix des moyens retenus pour développer le prototype sera essentiellement guidé par un souci de rapidité de réalisation et de flexibilité.
- par contre, le choix des moyens pour réaliser la version finale de l'application tiendra compte en général de préoccupations visant à réduire les coûts et les délais d'exploitation de l'application ainsi qu'à lui assurer un niveau de sécurité maximum.

I.3.3 - Quelles sont les applications pouvant justifier la construction d'un prototype ?

Puisqu'il est bien évident que toute conception d'une application informatique n'exige pas la construction d'un prototype, nous avons essayé de dégager une typologie de celles pour lesquelles un prototype peut être envisagé. Expérimentalement, nous avons été amenés à distinguer trois catégories d'applications :

a) les applications impliquant un *coût élevé et de longs délais de réalisation,* seront plus justement évaluées avec un prototype. Si en particulier, une décision d'arrêt du projet intervient après utilisation du prototype, cette décision peut certainement être mieux comprise et acceptée, car elle peut s'appuyer, non pas seulement sur des dossiers d'étude, mais sur les éléments concrets fournis par les tests exécutés sur le prototype.

b) un projet informatique apportant des possibilités nouvelles qui ne peuvent être exploitées que si l'organisme au profit duquel on le conçoit doit *modifier profondément ses méthodes de travail, restructurer ses services ou créer une nouvelle organisation.* Cet organisme doit alors imaginer une nouvelle forme d'activité de son organisation au moment où il collabore aux spécifications du projet. Cet effort est très difficile, et, pour le mener à bien, l'existence d'un prototype montrera concrètement les possibilités de l'application informatique et l'adéquation de l'environnement nécessaire à son bon fonctionnement.

c) *une application informatique "floue",* dans la mesure où parmi tous les services qu'elle peut rendre, le choix, forcément limité parmi eux, qu'elle suppose, est difficile. Les applications floues conduisent généralement à une conception souvent remise en cause et à un développement itératif très accentué. Pour une telle application, la construction d'un prototype devient preuve de faisabilité et surtout test d'adéquation des services rendus aux priorités retenues.

I.3.4 - Qui est concerné par la réalisation d'un prototype d'une application ?

Bien évidemment, ce sont en priorité les futurs utilisateurs du projet, mais aussi le responsable de ce projet et les informaticiens avec lesquels il le réalisera.

a) *les futurs utilisateurs* de l'application seront certainement les plus directement concernés. Bien évidemment, ils le sont parce que c'est par leur seule appréciation qu'est validé ou modifié le prototype. Il est prouvé expérimentalement que leur implication par un prototype dans un projet informatique, leur demande une quantité de travail importante mais que celle-ci est en général fournie parce qu'elle propose une activité plus concrète que les classiques discussions autour d'un dossier. La mise en oeuvre par ces utilisateurs d'un prototype qui fonctionne, modifie radicalement leur comportement : leur compétence en informatique leur paraît beaucoup moins essentielle car ils ont alors la possibilité d'affirmer et d'utiliser beaucoup plus nettement leurs compétences dans leur spécialité. Ils mesurent aussi beaucoup mieux la nécessité d'exprimer leurs desiderata d'une façon précise.

b) *le responsable du projet* est, quant à lui, assuré à la fin de l'analyse fonctionnelle d'avoir spécifié, de manière précise et relativement exhaustive, les diverses fonctions que devra assurer l'application puisque, dans le prototype, elles sont exécutables. Il aura, par ailleurs, obtenu ces spécifications par une décomposition en couches successives indépendantes. Il pourra, de ce fait, planifier en meilleure connaissance de cause la phase de réalisation technique.

Par ailleurs, avant que la version finale ne soit définitivement achevée, le responsable du projet pourra, avec le prototype, mieux organiser l'information, la formation et le recyclage des futurs utilisateurs en les plaçant dans des situations de travail concrètes (postes d'acquisition des données, correction des anomalies, utilisation des données produites en relation avec les décisions qu'elles supposent).

c) *les informaticiens chargés de la réalisation technique* pourront élaborer la construction de la version opérationnelle sans problèmes majeurs. Avec le prototype, ils disposent en effet, d'un véritable dossier d'analyse qui permet de :

- supprimer les données soit inutiles, soit trop coûteuses à acquérir, soit de valeur non garantie,
- réduire ou confirmer les redondances d'informations,
- identifier les données et spécifier les contrôles associés,
- évaluer le volume moyen des données et leur taux de mise à jour,
- tester les circuits d'acquisition de données (y compris les circuits d'erreurs) avec leurs postes de travail,
- valider les schémas logiques des traitements,
- étudier les relations entre les données, tout comme celles entre les traitements et les données,
- calculer des fréquences d'accès à des données, compte-tenu des transactions, pour déceler les points critiques de l'application qui devront faire l'objet de choix organiques particuliers dans les méthodes d'accès permettant des temps de réponse suffisants,
- guider les choix d'un langage de programmation et d'un système d'exploitation,
- évaluer les volumes de programmation et améliorer la productivité en programmation,
- faciliter les tests puisque des jeux d'essais ont déjà été produits par l'utilisation du prototype.

I.3.5 - Conditions techniques de réalisation d'un prototype

Compte-tenu des remarques précédentes, les moyens indispensables pour élaborer un prototype doivent permettre :

- une *réalisation rapide*, donc une très grande productivité au niveau de la programmation, pour l'obtenir dans des délais comparables à ceux constatés dans les méthodes classiques d'étude fonctionnelle d'un projet.

- une *réalisation facilement modifiable* ; la flexibilité nécessaire du prototype implique une flexibilité dans les méthodes de gestion des données comme dans celles de programmation.

- une *autodocumentation maximale* du prototype dans la mesure où la dernière version de celui-ci servira de cahier des charges pour la réalisation de la version opérationnelle du projet.

. *Une réalisation rapide* :

Ces contraintes de construction rapide d'un prototype supposent que soient disponibles des moyens logiciels importants.

Les expériences que nous avons pu réaliser personnellement ou dont nous avons eu connaissance permettent de dégager les caractéristiques que doivent apporter ces logiciels [L14] [L15] [A12] [A13] et [A14].

a) il faut tout d'abord disposer d'un langage de programmation de très haut niveau permettant l'allocation dynamique des variables en mémoire centrale, offrant des opérations très synthétiques pour favoriser une programmation modulaire. Par exemple, APL, Algol 68 ou le langage de la machine Alvan répondent approximativement à ces exigences.

b) il faut disposer, par ailleurs, d'un système de base de données qui supporte ce langage de programmation. Le système de gestion de base de données doit offrir certaines fonctions essentielles :
   - une gestion automatique des entrées-sorties sur la base sans obligation de définir un format pour les données,
   - un appel associatif des informations,
   - une structuration des données sous forme de réseau ou d'arbre,
   - une fonction de gestion de transaction avec un moniteur de télétraitement performant.

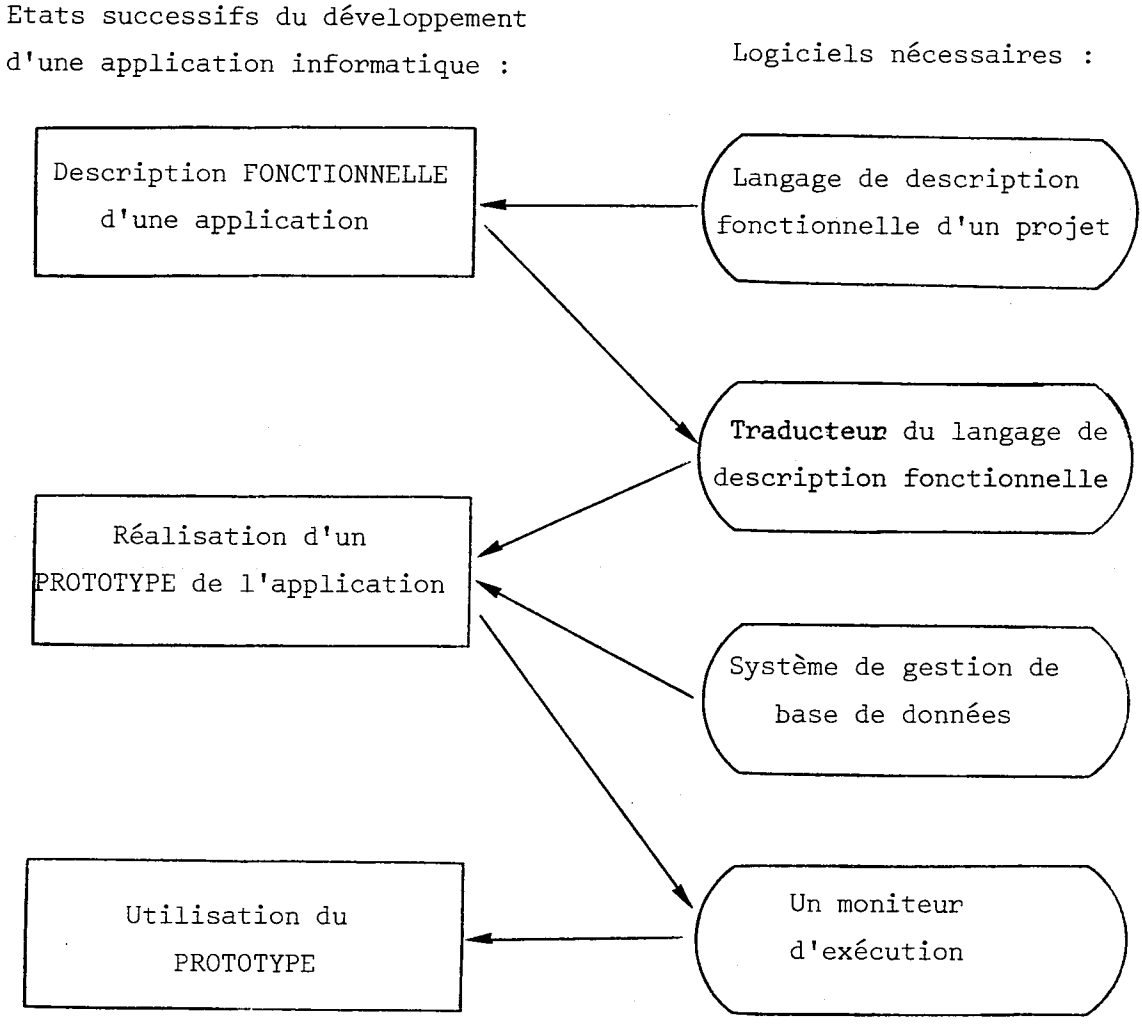
. *Une réalisation facilement modifiable et autodocumentée :*
L'expérience prouve aussi qu'un langage de programmation de haut niveau et un système de gestion de base de données ne sont pas suffisants pour avoir toute la flexibilité souhaitable qui permette une maintenance rapide du prototype. Ils imposent toujours, en effet, certains choix peu modifiables comme par exemple :
- les méthodes d'accès privilégiées aux données de la base,
- la distinction entre mode conversationnel ou traitement par lots,
- l'ordre d'introduction des informations en entrée ou de leur affichage en sortie.

Ces choix sont en contradiction avec l'obligation faite à un prototype de représenter la seule spécification fonctionnelle complète d'un projet, en laissant tous les degrès de liberté nécessaires pour simuler différents choix organiques.

I-4 - LES PROPOSITIONS MACSI-P

Nous en sommes arrivés à la conclusion qu'il fallait définir un *MODELE FONCTIONNEL D'APPLICATION* et spécifier le *LANGAGE* permettant de décrire une application selon ce modèle. Il fallait, par ailleurs, des *OUTILS LOGICIELS* qui permettent, à partir d'une description dans ce langage, de produire, de façon plus ou moins automatique, le prototype exécutable sur un certain système de base de données. Les travaux du groupe INFORSID 2 [L16] (conception et réalisation des systèmes d'informations automatisées) conduisent à une telle distinction entre modèle, langage et outils.

Le schéma suivant résume la séquence de mise en oeuvre de ces outils.



Nous avons ainsi défini[L22] [A18] [A19] un modèle et un langage et réalisé les outils logiciels correspondants sur le système SOCRATE [L23].

I.4.1 - Le modèle fonctionnel d'une application doit permettre de décrire pour une application :

a) *l'organisation générale de l'application informatique* par rapport à son environnement administratif de gestion. Une *APPROCHE MODULAIRE descendante* telle que F. PECCOUD la propose [L22] semble être une solution nécessaire si l'on pose comme objectifs que, d'une part, la définition des FONCTIONS de l'application précède le choix des ORGANES de réalisation et que, d'autre part, l'utilisateur potentiel de l'application a la maîtrise d'oeuvre des spécifications du système d'informations. Cette solution est une adaptation de certains concepts méthodologiques récents (modularité, analyse descendante) apparus dans le domaine de la programmation structurée [L6] [L7] et [L19].

Nous ne présenterons qu'un résumé (cf. annexe B) des principes fondamentaux de cette approche : F. PECCOUD [L22] et A. CHELMINSKI [L5] en ont développé une présentation beaucoup plus complète.

b) *les données*, leurs définitions et leurs relations logiques sans aucune espèce d'hypothèses sur les choix des méthodes d'accès ou de représentation en mémoire. Nous proposons (cf. chapitre II) *UN MODELE ENSEMBLISTE* pour décrire une structure fonctionnelle des données d'une application. Nous nous sommes personnellement inspirés [L22] [A18] des travaux de CODD [A9] , ABRIAL [A1] et DELOBEL [L9] et des rapports CODASYL [A7] [A8] (cf. annexe A).

c) *les transactions* en entrée ou en sortie, dans ce qu'elles ont de fonctionnellement spécifique indépendamment des moyens techniques de saisie, d'édition ou des modes de traitements (temps différé, temps réel conversationnel ou non, saisie intelligente décentralisée, ...).

Par caractéristiques fonctionnelles de ces transactions, nous entendons essentiellement :

- les données qu'elles manipulent,
- l'ordre dans lequel elles sont lues ou éditées,
- les contrôles associés et le niveau d'exécution des transactions auquel ils doivent intervenir,

. les sémantiques standards de création, suppression ou modification d'enregistrement qu'elles supposent implicitement.

Nous proposons (cf. chapitre II) d'utiliser *un schéma d'automates d'états finis* pour exprimer de façon claire et précise ces caractéristiques de transactions [L13] , [L14] , [A18] et [A22] .

I.4.2 - Le langage de description fonctionnelle doit par ailleurs avoir des contraintes, de nature syntaxique, qui obligent ceux qui s'en servent à systématiquement documenter, ne serait-ce qu'avec des commentaires, leur description fonctionnelle du projet (cf. chapitre III).

I.4.3 - Des outils logiciels produisent presque automatiquement le prototype à partir de spécifications conformes au modèle.

Dans l'état actuel de nos travaux, nous avons construit un logiciel (cf. chapitre IV et annexe C) qui, à partir d'une description selon notre modèle fonctionnel, permet de produire automatiquement environ 80 % des instructions du prototype (cf. § C12.4).

. il réalise une *traduction complète d'un schéma fonctionnel* des données en *une structure SOCRATE.*
. il traduit les schémas de description fonctionnelle des transactions en *produisant les programmes de lecture ou d'écriture* correspondants, dans lesquels sont incorporées les formes de contrôle les plus usuelles.
. il assure une fonction de moniteur qui permet l'exécution de ces transactions soit en conversationnel, soit en traitement par lots.

Au code SOCRATE généré automatiquement par ce logiciel, le réalisateur du prototype doit ajouter lui-même des sous-programmes SOCRATE qui implémentent des traitements très spécifiques du problème. Le superviseur assure au niveau de l'exécution l'appel de ces sous-programmes au bon moment.

I.4.4 - Plan de cet ouvrage

Nous avons essayé de structurer cet ouvrage pour qu'il puisse être lu par des personnes dont les besoins diffèrent. Ainsi ce document peut constituer :

- *un manuel de présentation* de modèles d'aide à la conception de systèmes d'informations ; ce sont les chapitres :

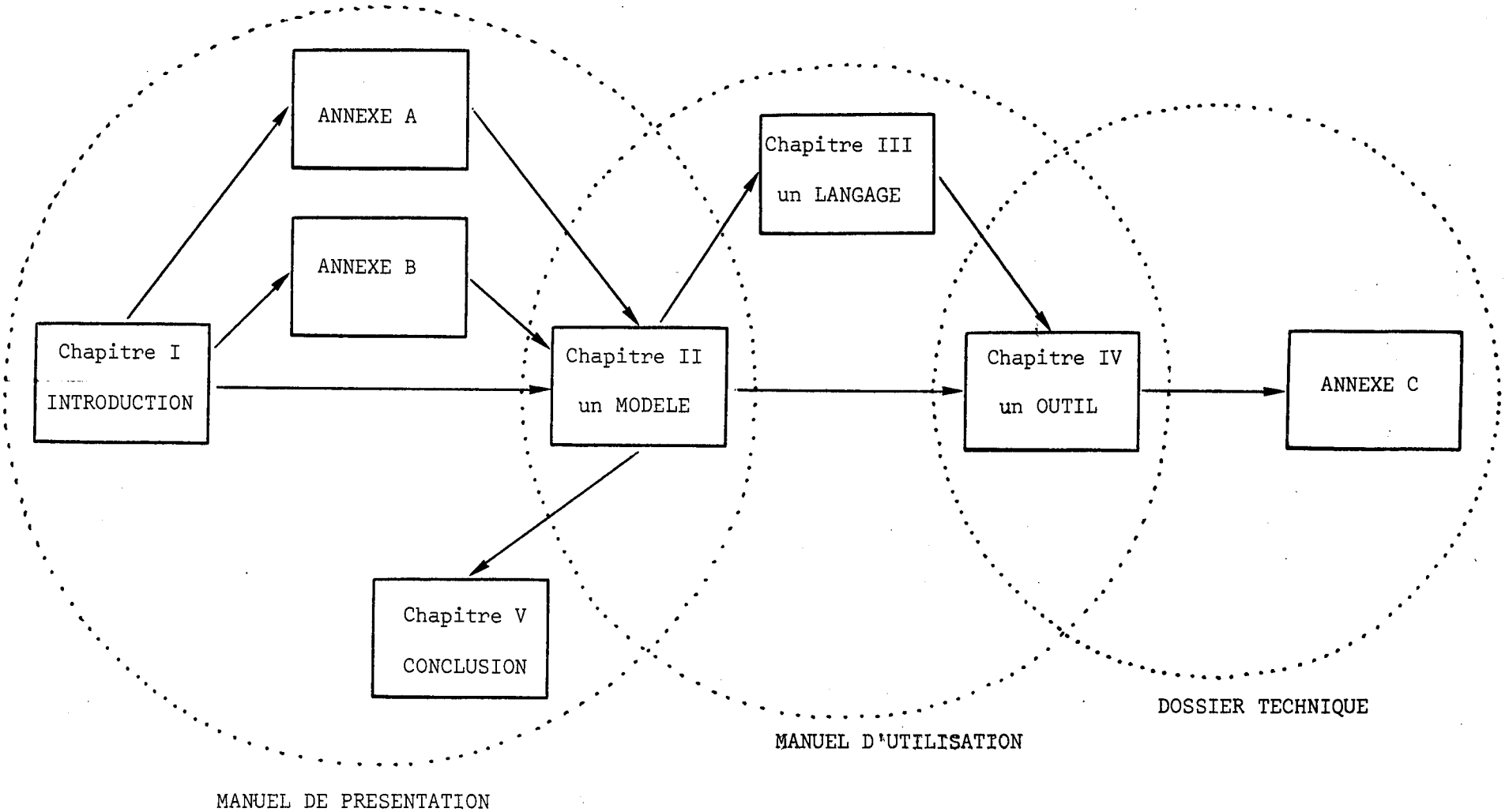
  - chapitre I : introduction
  - chapitre II : un modèle fonctionnel d'application
  - chapitre V : conclusions
  - annexe A : présentation de quelques modèles de systèmes d'informations
  - annexe B : le projet MACSI-1.

- *un manuel d'utilisation* du logiciel MACSI-P :

  - chapitre II : un modèle fonctionnel d'application
  - chapitre III : un langage de description
  - chapitre IV : un outil : le logiciel MACSI-P.

- *un dossier technique* destiné à des informaticiens qui ont la charge de réaliser des prototypes d'applications informatiques :

  - chapitre IV : un outil : le logiciel MACSI-P
  - annexe C : réalisations techniques.



ARTICULATION DES CHAPITRES

CHAPITRE II

UN MODÈLE FONCTIONNEL D'APPLICATION

II-1 - INTRODUCTION

II.1.1 - Choix d'un exemple pour introduire tous les concepts du modèle

Nous illustrerons le modèle que nous voulons présenter avec une seule application dont il faut esquisser préalablement les contours.

Cet exemple décrit en partie une application ayant pour objectif de gérer sur ordinateur les problèmes liés à la scolarité dans un département d'IUT, tels que nous avons pu les connaître et tels que le lecteur peut facilement les deviner.

Nous nous intéresserons essentiellement aux activités du secrétariat qui doit :

a) chaque année,

- inscrire des étudiants dans chacune des deux années d'études et constituer des dossiers-étudiants ;
- répartir les étudiants en groupes de travaux pratiques (TP), travaux dirigés (TD), cours ;
- coordonner l'activité des enseignants du département, qu'ils soient titulaires ou vacataires en répartissant leurs services d'enseignement ;
- distribuer au début de l'année le programme pédagogique annuel discuté et modifié lors de réunions des enseignants au cours de l'année précédente ;
- affecter les locaux disponibles pour les cours et séances d'exercices ;
- en fin d'année, vérifier que le programme pédagogique a été assuré.

b) chaque trimestre,

- établir un emploi du temps-type conforme aux indications du programme pédagogique et aux voeux des enseignants ;
- assurer la réservation des salles nécessaires ;
- faire un bilan, en fin de trimestre, pour mesurer l'écart entre ce qui était prévu et ce qui a réellement été fait.

c) et chaque semaine,

- prendre en compte diverses modifications définitives ou temporaires de l'emploi du temps : comme par exemple la prévision

d'une semaine bloquée de devoirs surveillés ou d'enseignements spéciaux, le rattrapage des heures perturbées par des grèves, l'impossibilité d'utiliser des salles en réparation ;
- éditer l'emploi du temps hebdomadaire complet indiquant les heures d'enseignements, les groupes, les salles, les enseignants et les matières concernées ;
- fournir à chaque enseignant, comme à chaque groupe d'élève, son propre emploi du temps.

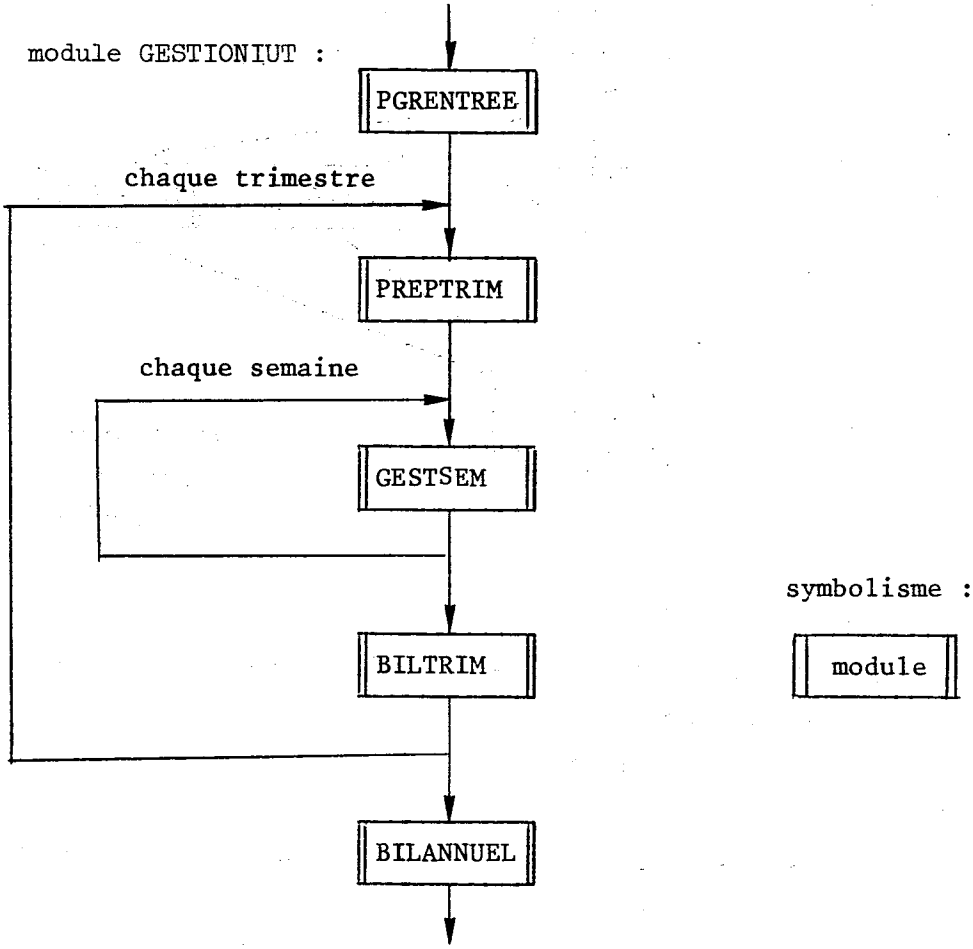
Nous supposons pour des raisons de simplicité que le choix d'une organisation de l'emploi du temps sera fait manuellement par le secrétariat. Cette hypothèse exclut l'usage d'un programme d'ordonnancement sous contrainte permettant le calcul automatique de l'emploi du temps ; nous savons que certains existent [cf. L12].

Pour mieux définir le cadre de cet exemple, nous proposons d'abord au lecteur une décomposition modulaire de cette application conforme aux principes MACSI-1 [L22] [L5]résumés en Annexe B, de façon à donner son architecture générale.

II.1.2 - Décomposition modulaire de l'application

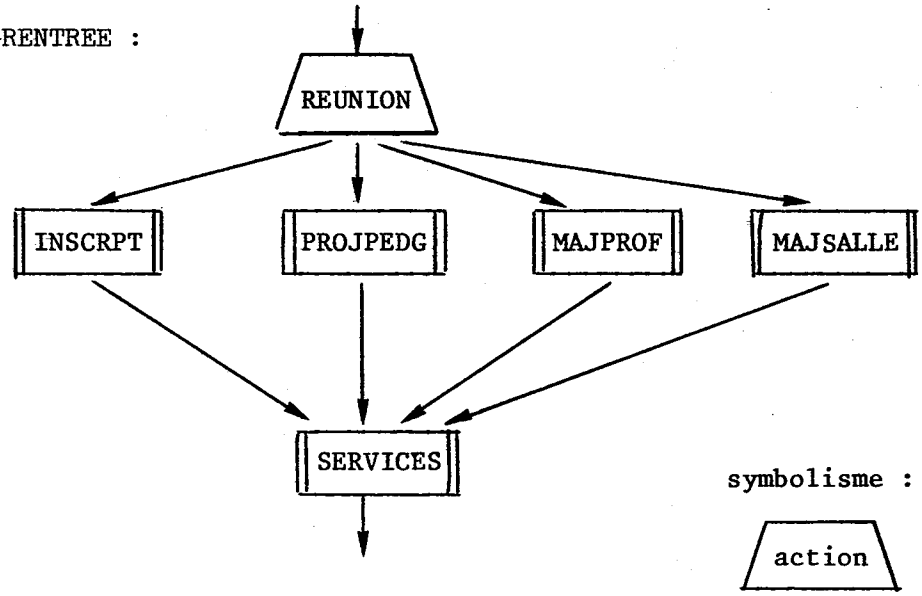
Toute l'application elle-même est considérée comme un MODULE ayant pour code GESTIONIUT.

L'entrée unique du module GESTIONIUT est le module PGRENTREE, sa sortie BILANNUEL.

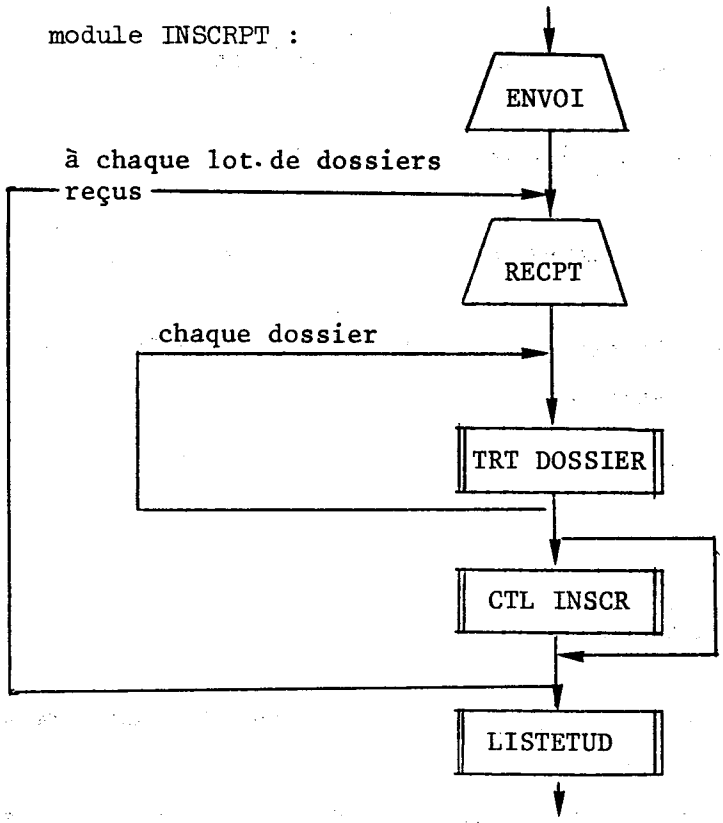



```
defmod GESTIONIUT * gestion d'une partie de la scolarité dans un département
                    d'IUT *
    m PGRENTREE * préparation générale de la rentrée *
    tantque * le dernier trimestre de l'année en cours n'est pas terminé *
          faire
              m PREPTRIM * préparation du trimestre *
              tantque * le trimestre en cours n'est pas terminé *
                    faire
                        m GESTSEM * gestion de l'emploi du temps de la
                                    semaine *
                    refaire
              m BILTRIM * bilan trimestriel *
          refaire
    m BILANNUEL * bilan annuel *
fmod
```

module PGRENTREE :




```
defmod PGRENTREE
|   a REUNION * réunion du conseil de direction décidant du démarrage
|               des préparatifs généraux de la rentrée *
|     par DICO * conseil de direction *
|   indep
|   |   m INSCRPT * inscription des étudiants dans les différentes années
|   |               d'études *
|   |   m PROJPEDG * mise à jour du projet pédagogique *
|   |   m MAJPROF * mise à jour de la liste des enseignants pour l'année
|   |               en cours *
|   |   m MAJSALLE * recensement des locaux disponibles *
|   findep
|   m SERVICES * élaboration des divers services des enseignants *
fmod
```




```
defmod INSCRPT
    a ENVOI * envoi de dossiers d'inscription *
      par SECRT * secrétariat de la scolarité *
    tantque * on reçoit des dossiers et date limite non dépassée *
      faire
        a RECPT * réception des dossiers d'inscription *
          par SECRT
        tantque * tous les dossiers ne sont pas encore traités *
              faire
                  m TRTDOSSIER * gestion d'un dossier d'inscription *
              refaire
        si * à la demande du conseil de direction *
          alors
              m CTLINSCR * contrôle de l'état des inscriptions *
        fsi
      refaire
    m LISTETUD * édition des listes de tous les étudiants inscrits *
fmod
```

```
defmod CTLINSCR
|    t TSLISTP * édition des listes partielles d'inscription *
|      par SECRT
|    a BILINSCR * bilan des inscriptions en cours *
|      par DICO
|    si * inscriptions insuffisantes et avant le 15 septembre *
|    | alors
|    |      m RELINSCR * relancer les inscriptions par des envois de
|    |                   dossiers *
|    finsi
fmod
```

```
defmod LISTETUD
|    a FININSCRPT * clôture des inscriptions *
|      par DICO
|    t TSETUD1 * édition de la liste des étudiants inscrits en 1ère année *
|      par SECRT
|    t TSETUD2 * édition de la liste des étudiants inscrits en 2ème année *
|      par SECRT
|    a ENVLIST * envoi des listes des étudiants aux enseignants *
|      par SECRT
fmod
```

```
defmod TRTDOSSIER
|    t TEINSCR * inscription d'un étudiant *
|      par SECRT
|    a CLTDOS * contrôle de l'archivage du dossier d'un étudiant inscrit *
|      par SECRT
fmod
```

```
defmod MAJPROF
|   a MODPRF * recenser les modifications du corps professoral *
|     par SECRT
|   tantque * le secrétariat doit modifier la liste des enseignants pour
|             l'année en cours *
|           faire
|           |   si * arrivée d'un nouvel enseignant titulaire ou vaca-
|           |   |      taire *
|           |   |  alors t TECREPROF * ajout d'un nouvel enseignant *
|           |   |                      par SECRT
|           |   fsi
|           |   si * départ d'un enseignant titulaire ou refus d'un en-
|           |   |      seignant vacataire de continuer des vacations *
|           |   |  alors t TESUPROF * suppression d'un enseignant *
|           |   |                     par SECRT
|           |   fsi
|           |   si * modifications des caractéristiques d'un enseignant
|           |   |      (statut, mi-temps, ...) *
|           |   |  alors t TEMODPROF * modifications des renseignements
|           |   |                      relatifs à un enseignant *
|           |   |                      par SECRT
|           |   fsi
|           refaire
|   t TS LISTPROF * édition de la liste des enseignants *
|     par SECRT
fmod
...etc...
```

Avertissement :

Tous les exemples cités dans la suite de ce chapitre permettront de décrire fonctionnellement des DONNEES et des TRANSACTIONS de cette application dont l'esquisse vient d'être dessinée par cette décomposition modulaire. Mais nous ne proposerons pas une solution complète pour ce projet : une ébauche de solution, par sa concision, sera un meilleur outil de présentation du modèle MACSI-P.

II-2 - APPROCHE STATIQUE DU MODELE

II.2.1 - Les ensembles de base et leurs propriétés

II.2.1.1 - Les ensembles de base

Définition :

> **Les ensembles de base sont des ENSEMBLES d'ELEMENTS CONCRETS dont l'existence est évidente dans l'univers physique que décrit l'application informatique envisagée.**

Ces ensembles de base regroupent des éléments du monde réel, IDENTIFIABLES INDIVIDUELLEMENT ; ces éléments ont une vie : ils apparaissent, évoluent, disparaissent et la gestion, pour laquelle est construite l'application informatique, s'appuie sur une connaissance de la situation de chaque élément.

CODE et LIBELLE d'un ensemble :

Chaque ensemble de base est

- désigné par un code d'identification discriminant appelé *CODE de l'ENSEMBLE* qui est souvent le nom commun qualifiant un élément de cet ensemble : le "CODE" de l'ensemble permet donc de désigner l'un de ses éléments comme l'ensemble lui-même ;

- décrit par une chaîne de caractères, appelée *LIBELLE de l'ENSEMBLE*, expliquant son contenu.

Exemple 2.1 :

ETUDIANT, SALLE, ENSEIGNANT et MATIERE sont les codes des quatre ensembles de base dont nous donnons les libellés :

. *ETUDIANT = ensemble des étudiants d'un département d'IUT*
. *SALLE = ensemble des locaux d'un département d'IUT*
. *ENSEIGNANT = ensemble des personnes qui enseignent dans un département d'IUT*
. *MATIERE = ensemble des matières enseignées dans un département d'IUT.*

Ainsi chaque étudiant, chaque salle, chaque enseignant, chaque matière existe bien concrètement ; il est identifiable et doit être distingué de tous les autres éléments.

La gestion scolaire consiste à, entre autres opérations, pour les étudiants :

- *créer des étudiants → inscription*
- *faire évoluer des étudiants → formation*
- *faire disparaître des étudiants → délivrance du diplôme à la fin de la scolarité, décès, abandon des études.*

pour les salles :

- *créer des salles → nouvelles constructions*
- *modifier des salles → travaux d'entretien dans les locaux, équipements spécialisés*
- *supprimer des salles → déclaration de locaux insalubres ou ne répondant pas aux normes de sécurité.*

Liste des propriétés d'un ensemble

Chaque ensemble de base, désigné par son CODE, expliqué par son LIBELLE, est caractérisé par la *LISTE de ses PROPRIETES* (voir définition ci-après).

II.2.1.2 - Les PROPRIETES d'un ENSEMBLE

Définition :

Une propriété P d'un ensemble E est une application de cet ensemble E sur un espace de valeurs V

$$P : E \rightarrow V$$

Code, libellé et espace de valeurs d'une propriété

Une propriété est définie par :

- son *CODE*, qui est le nom (P) de l'application ; il doit être discriminant afin de l'identifier sans ambiguïté ;
- son *LIBELLE* explicatif ;
- son *ESPACE de VALEURS*(l'espace de valeurs (V) associé à l'application).

Exemple 2.2

CODE d'un ENSEMBLE de BASE	*PROPRIETES d'un ENSEMBLE*		
	CODE	*LIBELLE*	*ESPACE de VALEURS*
ETUDIANT	*NOM*	*nom d'un étudiant*	*chaîne alphabétique avec les caractères spécifiant "-" ou "'"*
	PRN	*prénom d'un étudiant*	*- idem -*
	NUMETUD	*numéro de sa carte d'étudiant*	*numéro de 6 chiffres*
	ANNAIS	*année de naissance*	*de 1945 à 1965*
	INSEE	*numéro INSEE*	*chaîne numérique de 13 caractères..*
	ADRESSE	*adresse personnelle de l'étudiant*	*chaîne alphanumérique*
SALLE	*CDS*	*nom d'une salle*	*chaîne alphanumérique de 4 caractères*
	CDBAT	*nom du bâtiment*	*{a,b,c,d}*
	ETAGE	*numéro de l'étage*	*de 1 à 5*
	CAPACITE	*nombre de places assises*	*de 1 à 500*

Les divers espaces de valeurs V ainsi associés aux ensembles pourraient être appelés ENSEMBLES ABSTRAITS, car ils n'ont aucune existence physique et n'apparaissent dans un système d'informations que par l'intermédiaire de faits qui les associent aux éléments concrets des ensembles de base.

Notation d'un ensemble de base

Un ensemble de base de code E et ses n propriétés (de code P1, P2, ... Pn) seront notés :

E [P1, P2, ... Pi, ... Pn]

ou d'une façon plus précise :

E [P1:V1, ... Pi:Vi, ..., Pn:Vn]

Cette notation pour décrire des ensembles et des propriétés est utilisée pour alléger la présentation des différentes définitions, des règles méthodologiques et des exemples associés qui constituent l'essentiel de ce chapitre II.

Notation d'un élément d'un ensemble

La description de chaque élément e d'un ensemble de base E [P1,P2,...Pn] se fait en évaluant les VALEURS de PROPRIETES de l'ENSEMBLE ; il est représenté par un n-uplets de valeurs :

$$e = (P1(e), P2(e), \ldots Pn(e))$$

Exemple 2.3

Ensemble de base : ETUDIANT [NOM, PRN, INSEE, ...]
éléments : (Dubois, Jean, 1490538130030, ...)
(Durand, Paulette, 2470369196176, ...)

Ensemble de base : SALLE [CDS, CDBAT, ETAGE, CAPACITE, ...]
éléments : (C211, c, 2, 40, ...)
(C208, c, 2, 40, ...)
(b309, b, 3, 5, ...)

Valeurs particulières de l'espace de valeurs de toute propriété

En plus des valeurs DEFINIES, appartenant effectivement à l'espace des valeurs d'une propriété quelconque, nous considérons deux valeurs particulières qui sont systématiquement à prendre en compte et qui font partie implicitement de ces espaces :

. la valeur U : qui signifie "Undefined" (valeur indéfinie) ou "Ultérieurement précisée".
Cette valeur est utilisée lorsqu'on désire indiquer qu'à un moment déterminé, cette information est inconnue mais qu'elle devra être définie plus tard.

. la valeur RAS : qui signifie "Rien A Signaler" ou valeur nulle, pour indiquer que cette valeur ne peut pas être définie pour un élément.

Unicité de la valeur

Toute propriété d'un ensemble a toujours *une valeur unique* (éventuellement une valeur particulière : 'U' ou RAS') pour un quelconque de ses éléments à un instant donné.

Si en apparence, une propriété P peut avoir plusieurs valeurs au même moment, on doit alors définir plusieurs propriétés P_1, P_2, ... P_k représentatives de ces valeurs.

Exemple 2.4

S'il faut prendre en compte trois valeurs possibles du prénom d'un ETUDIANT, on définit alors trois PROPRIETES :

PRN1 : premier prénom
PRN2 : deuxième prénom
PRN3 : troisième prénom

ETUDIANT [NOM, PRN1, PRN2, PRN3, ...]
(Dupont, Jean, Hugues, Antonin, ...)
(Durand, Paulette, Maris, RAS, ...)
(Dubois, Jean-Pierre, RAS, RAS, ...)
(Duroc, U, ...)

Unicité de l'appartenance d'une propriété à un ensemble

Une propriété est ATTACHEE EXCLUSIVEMENT A UN ENSEMBLE et à un seul.

C'est-à-dire que deux ensembles de base ne peuvent pas avoir, par définition, de propriétés communes. Nous verrons plus loin l'intérêt de cette convention aux paragraphes § II.2.1.5 et § II.2.2.4 .

Exemple 2.5

Soient trois ensembles distincts : ETUDIANT, SALLE et ENSEIGNANT ; tous les éléments de ces trois ensembles ont un "NOM", mais par respect de la convention précédente, cette propriété aura un code d'identification différent suivant les ensembles, et des espaces de valeurs distincts. On peut remarquer ainsi que le "NOM d'une SALLE" est sémantiquement différent du "NOM d'un ETUDIANT". On distinguera ces trois propriétés en leur donnant des codes distincts.

ETUDIANT [NOM, ...]
SALLE [CDS, ...]
ENSEIGNANT [NOMP, ...]

Remarque : Quand nous disons "PROPRIETES d'un ENSEMBLE", nous commettons un abus de langage : il faudrait parler de "PROPRIETES des ELEMENTS d'un ENSEMBLE". Dans d'autres ouvrages, on trouve diverses dénominations (caractéristiques, items, constituants, attributs, rubriques, etc ...) qui recoupent cette notion de propriété.

II.2.1.3 - Les propriétés-clés d'un ensemble

Des éléments distincts d'un même ensemble ont toujours des n-uplets qui diffèrent par au moins une valeur. Les propriétés d'un ensemble E [P1,P2,...Pn] sont telles que :

$\forall e, e' \in E$ et $e \neq e'$, les n-uplets (P1(e), P2(e), ... Pn(e)) et (P1(e'), P2(e'), ... Pn(e')) sont distincts.

Définition :

Une PROPRIETE-CLE d'un ensemble E est une propriété C de cet ensemble E, définie comme une APPLICATION INJECTIVE de E sur l'espace de valeurs V de C.

E [C, P1, P2, ... Pn], C propriété-clé de E

$\forall e, e' \in E, e \neq e' \implies C(e) \neq C(e')$

Une telle propriété permet de distinguer sans ambiguïté, à un moment donné, tout élément d'un ensemble, car des éléments quelconques n'ont pas la même valeur pour cette propriété : chaque valeur discriminante est "la clé d'un élément".

Propriété-clé obligatoire

Tout ensemble de base doit avoir au moins une propriété-clé.

Celle-ci ne sera pas toujours unique.
Ainsi tout élément d'un ensemble est identifiable sans ambiguïté par la valeur discriminante d'une des propriétés-clés de l'ensemble.

Notation d'une propriété-clé d'un ensemble

On soulignera d'un trait continu la ou les propriétés-clés d'un ensemble E.

E [P1, P2, ... Pi, ... Pk, ... Pn]

Exemple 2.6

L'ensemble de base SALLE a une seule propriété-clé (CDS : nom de la salle) alors que l'ensemble ETUDIANT en possède deux (INSEE : numéro INSEE, NUMETUD : numéro de sa carte d'étudiant).

ETUDIANT [*NOM, PRN, INSEE, ..., NUMETUD, ...*]
SALLE [*CDS, CDBAT, ETAGE, CAPACITE, ...*]

$\forall e1, e2 \in$ *ETUDIANT et* $e1 \neq e2$ *INSEE (e1)* $\neq$ *INSEE (e2)*
NUMERO (e1) $\neq$ *NUMERO (e2)*

$\forall s1, s2 \in$ *SALLE et* $s1 \neq s2$ *CDS (s1)* $\neq$ *CDS (s2)*

Remarque méthodologique

Il existe de nombreux cas d'ensembles de base où un même élément est identifiable par différentes propriétés suivant l'application étudiée.

Une personne est identifiée par :

- son numéro INSEE dans le cas de la sécurité sociale,
- son numéro de contribuable pour la gestion de ses impôts,
- son numéro de contrat EDF pour une facturation de sa consommation en électricité,
- son numéro de sociétaire MAIF pour ses contrats d'assurance à cette mutuelle,

...

Aussi, au cours de l'étude d'une application, avant de définir une propriété-clé des éléments d'un ensemble, faut-il analyser les clés éventuelles que ces éléments possèdent déjà dans d'autres applications. La décision de créer une nouvelle clé ne doit être prise que si les autres clés sont jugées :

- inadéquates (un enseignant n'a un numéro de sociétaire MAIF que s'il a au moins un contrat d'assurance avec cette mutuelle) ;
- difficiles à obtenir (NUMERO INSEE pour les étrangers) ;
- trop longues à utiliser (NUMERO INSEE sur 13 caractères remplacé par un code de 6 caractères pour la gestion de la scolarité) ;
- insuffisantes dans la mesure où la création d'une nouvelle propriété-clé est souvent le résultat d'une concaténation de propriétés favorisant certains traitements (comme l'intégration de la position géographique actuelle de l'intéressé dans un code de contribuable ou un code de contrat EDF).

Souvent, plusieurs clés d'un même élément sont enregistrées dans une application. Ainsi un service de gestion de la scolarité enregistre pour tout ETUDIANT son NUMERO d'étudiant et son NUMERO INSEE.

II.2.1.4 - Extensions sémantiques du modèle des données

Dans le modèle présenté, la description d'un nouvel ensemble de base doit contenir obligatoirement son CODE, son LIBELLE, la liste de ses PROPRIETES et ses PROPRIETES-CLES. Ceci n'est pas toujours suffisant pour décrire sans ambiguïté un ensemble.

Définition et taille d'un ensemble

Pour améliorer la caractérisation d'un ensemble, nous complétons une telle description par :

- une DEFINITION précise de l'ensemble en COMPREHENSION ou en EXTENSION ;
- une TAILLE indiquant le nombre d'éléments que l'ensemble de base aura *au plus* à un instant donné, pendant la période pour laquelle l'application informatique est réalisée.

Exemple 2.7 :

Soit l'ensemble de base :
CODE : ETUDIANT
LIBELLE : ensemble des étudiants d'un département d'IUT
PROPRIETES : NOM, PRN, INSEE, NUMETUD
PROPRIETES-CLES : INSEE, NUMETUD

Cet ensemble est précisé par :
DEFINITION : ensemble des étudiants inscrits avant le 15 septembre de l'année en cours, dans un département d'IUT
TAILLE : 500.

Difficultés et importance de la DEFINITION d'un ensemble

On peut préciser un ensemble *en COMPREHENSION* si on sait donner une définition, sous forme de texte libre, qui indique, avec la plus extrême précision, les critères qui permettent de savoir si un élément appartient ou non à l'ensemble. Ceci n'est pas toujours facile.

Exemple 2.8 :

Ainsi dans l'exemple précédent, quelle définition en compréhension doit-on choisir pour l'ensemble des ETUDIANTS ?

- *Est-ce "les étudiants inscrits avant le 15 septembre" ?*
  (ainsi les étudiants inscrits avant le 15 septembre, mais qui ne se présenteront jamais au département car ils ont finalement choisi d'autres études, appartiennent à l'ensemble).
- *Est-ce les "étudiants inscrits et présents à tous les cours et TP" ?*
  (les étudiants inscrits en cours d'année appartiennent-ils à l'ensemble ?).
- *Est-ce "les étudiants qui passent les examens" ?*
  (le cas des auditeurs libres est-il alors prévu ?)

ou l'union de ces trois définitions ?

A défaut d'une possibilité de définition correcte d'un ensemble en compréhension, il faut le définir en extension.

La définition *en EXTENSION* d'un ensemble est la liste de tous ses éléments. Une telle définition n'est pratiquement pas utilisée pour des ensembles de plus de quelques dizaines d'éléments.

Exemple 2.9 :

Soit l'ensemble des MATIERES ; il est défini en extension par :
MATIERE = {mathématiques, programmation, analyse, gestion, anglais, techniques d'expression}.

TAILLE d'un ensemble

La TAILLE d'un ensemble ne peut être précisée qu'une fois explicitée la DEFINITION de l'ensemble et le champ de l'application informatique.

Exemple 2.10 :

Ainsi pour définir la taille de l'ensemble ETUDIANT, il faut préciser pour l'application informatique "gestion de la scolarité", s'il s'agit de la :
- *gestion des étudiants du département informatique de l'IUT de Grenoble, pour l'année scolaire en cours ?*
- *gestion des étudiants d'un département de l'IUT ?*
- *gestion des étudiants d'un département de l'IUT sur plusieurs années scolaires ? (problème de la constitution de fichiers historiques)*
- *gestion des étudiants de l'IUT 2 ?*

...

Aussi une bonne définition du domaine d'utilisation de l'application est-elle nécessaire pour donner une valeur à la taille d'un ensemble.

Remarques méthodologiques

- Cette taille doit donc tenir compte également des *extensions futures* de l'application : par exemple, la création d'une section "Année spéciale" ou d'un cycle "formation permanente" peut doubler les effectifs dans un département d'IUT.
- La taille permet de poser le problème des valeurs des propriétés-clés pour codifier tous les éléments d'un ensemble de base.

DEFINITION d'une propriété

En plus du CODE et du LIBELLE, la description d'une propriété peut être améliorée par une DEFINITION de son espace de valeurs en compréhension ou en extension (cf. définition d'un ensemble).

Exemple 2.11 :

La propriété NOM (nom d'un étudiant) peut être définie en compréhension : ... c'est une chaîne de caractères alphabétiques ou spéciaux comme " - " et " ' " ...

La définition du nom du bâtiment (CDBAT) où se trouve une salle est faite en extension :

$$CDBAT(s) \in \{a, b, c, d\}, \forall s \in SALLE.$$

Propriété composée

Définition :

> Une propriété d'un ensemble est une PROPRIETE COMPOSEE de plusieurs propriétés de cet ensemble si pour tout élément, la valeur de cette propriété composée est obtenue par CONCATENATION des valeurs des propriétés composantes pour le même élément.

Une propriété qui n'est pas composée sera dite *PROPRIETE ELEMENTAIRE.*

Notation d'une propriété P composée des propriétés P_i, P_j et P_k

$$P : (P_i \parallel P_j \parallel P_k)$$

Exemple 2.12 :

Ainsi, dans l'ensemble ETUDIANT, nous pouvons décrire trois propriétés composées :
INSEE : (SEXE || ANNEE || MOIS || DEPT || COMMUNE || N-ORDRE)
NUMETUD : (AN || NUMERO)
ADRESSE : (RUE : (NRUE || NOMRUE) || VILLE || BDIST : (CDPOST || BUREAU))
et dans l'ensemble SALLE :
CDS : (CDBAT || ETAGE || NUMERO).

Remarques méthodologiques

a) Il est possible qu'une propriété, élémentaire ou composée d'un ensemble, soit :
- propriété-clé de cet ensemble,
- propriété composante d'une ou plusieurs propriétés composées de cet ensemble.

b) Quand doit-on écrire une propriété composée ?
Nous conseillons la description d'une propriété composée Pi : (Pi_1 || Pi_2 || ... || Pi_k) lorsqu'il est possible de donner une réponse affirmative à l'une des questions suivantes :
- les spécifications des propriétés composantes Pi_j permettent-elles d'affiner la définition de Pi ? (le numéro INSEE est bien défini par ses composantes)
- existe-t-il des traitements sur la propriété Pi et sur au moins une composante Pi_j ? (adresse : (rue || ville || cdpost || bureau), adresse : utilisée dans un programme d'édition sans distinction de ses composantes, cdpost : utilisé dans un tri avant édition).
- en apparence, la propriété Pi peut-elle avoir plusieurs valeurs au même moment pour un élément ? (ce qui est contraire à la convention § II.2.1.2) ; les propriétés Pi_1, Pi_2, ... Pi_k sont alors représentatives de ces valeurs (exemple : PRENOM : (PRN1 || PRN2 || PRN3)).

II.2.1.5 - Conclusions provisoires

Après ces quatre définitions :
1 - ensemble de base
2 - propriété
3 - propriété-clé
4 - propriété composée,

nous pouvons recenser dans un tableau les caractéristiques principales des ensembles de base et des propriétés :

pour chaque :	caractéristiques fondamentales	caract. qualitatives
ENSEMBLE de BASE	- un CODE - une ou plusieurs PROPRIETES-CLES - des PROPRIETES	- LIBELLE - DEFINITION - TAILLE
PROPRIETE	- un CODE - un ESPACE de VALEURS - une COMPOSITION éventuelle	- LIBELLE - DEFINITION

Remarques méthodologiques sur l'utilité d'une codification unique d'une donnée :

Le modèle de données proposé ci-dessus permet d'identifier toute donnée relative à un élément d'un ensemble par un CODE UNIQUE DE PROPRIETE (une propriété est une caractéristique exclusive d'un ensemble).

Ce qui permet d'éviter les difficultés rencontrées dans d'autres modèles où le repérage d'une donnée est réalisé, en général, par une citation développant la hiérarchie structurelle des données définie lors de la description de la structure fonctionnelle.

Ce résultat permet d'envisager :

1 - un formalisme simple de description des entrées et des sorties (cf.§II.2.3)

2 - mais aussi des changements de codifications ou des codifications propres à chaque utilisateur, sans pour autant modifier la description structurelle des informations ni les programmes qui lui sont attachés.

Développons brièvement ce deuxième point :
pour chaque utilisateur potentiel du système d'informations, il serait possible de constituer une table de codification des données permettant d'assurer la correspondance bijective entre les codes définis dans la structure fonctionnelle des données selon le modèle proposé et les codes que l'utilisateur voudrait réellement utiliser.

Exemple 2.12 bis :

Exemple de table pour un utilisateur :
Table-k

CODES de l'UTILISATEUR	*CODES de la STRUCTURE FONCTIONNELLE*
NUMERO-INSEE	*INSEE*
TELEPHONE	*TEL*
ADRESSE-PERSONNELLE	*ADR1*
ADRESSE-PARENTS	*ADR2*
NOM	*NOM*
...	...

Naturellement la discriminance des codes doit être assurée dans chaque colonne, mais aussi entre colonnes pour éviter toute ambiguïté d'identification d'une donnée :

- $\forall\ i \neq j$ *TABLE-k* $(i,1) \neq$ *TABLE-k* $(j,1)$
  et *TABLE-k* $(i,2) \neq$ *TABLE-k* $(j,2)$
- $\forall\ i\ \nexists\ j \neq i$: *TABLE-k* $(i,1) =$ *TABLE-k* $(j,2)$.

II.2.2 - Les relations entre ensembles et leurs propriétés

II.2.2.1 - Les relations n-aires

Définition :

Une relation n-aire entre des ensembles de base est une PARTIE DU PRODUIT CARTESIEN de ces ensembles : elle correspond à un ensemble de n-uplets.

Soient n ensembles E, F, ... G et une relation n-aire R

$$R \subset E \times F \times \ldots \times G$$

alors, chaque élément r de la relation R est un n-uplet

$$r = (e, f, \ldots g) \text{ tel que } e \in E, f \in F, \ldots \text{ et } g \in G$$

Les ensembles E, F, ... G sont appelés *les ARGUMENTS* de la relation R ; le nombre des arguments est *le DEGRE* de la relation.

Les relations binaires (relations de degré 2) sont des cas particuliers de relations n-aires.

CODE, LIBELLE, DEFINITION, TAILLE et PROPRIETES d'une relation

Nous considérons que les relations sont de nouveaux ensembles que nous caractérisons comme les ensembles de base (cf. conventions § II.2.1.1, § II.2.1.4) par :

- un CODE d'identification discriminant,
- un LIBELLE explicatif,
- une DEFINITION précise,
- une TAILLE indiquant le nombre des n-uplets,
- une liste de PROPRIETES élémentaires ou composées.

Notation d'une relation

Nous pouvons adopter une notation condensée équivalente à celle employée pour les ensembles de base.

Une relation de code R d'arguments E_1, E_2, ... E_n et K propriétés P_1, P_2, ... P_k sont notés :

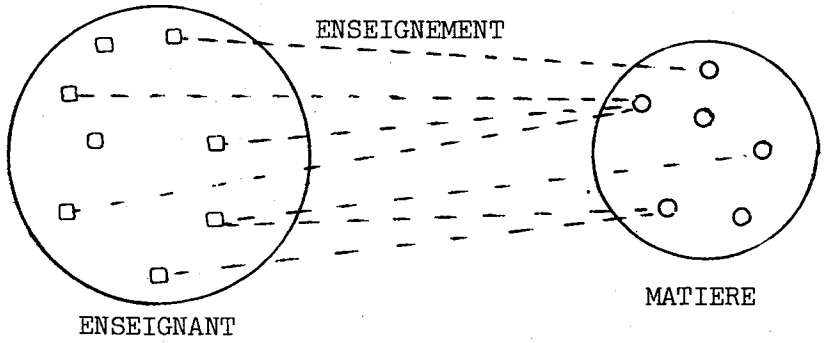
$$R\ (E_1, E_2, \ldots E_n)\ [P_1, P_2, \ldots P_k]$$

Exemple 2.13 :

Considérons successivement les relations ENSEIGNEMENT et SEANCE :

. ENSEIGNEMENT (ENSEIGNANT, MATIERE) [NIVEAU, TYPE, ...]

La relation ENSEIGNEMENT est une relation binaire entre les ensembles-arguments ENSEIGNANT et MATIERE ; elle regroupe des couples qui correspondent aux activités pédagogiques des enseignants d'un département de l'IUT.



	LEGENCE	*EXEMPLES*
□	*élément de l'ensemble ENSEIGNANT*	*Poincaré, Wirth, Knuth, ...*
○	*élément de l'ensemble MATIERE*	*mathématique, programmation, .*
----	*élément de la relation ENSEIGNEMENT*	*(Poincaré, mathématiques)*
		(Wirth, programmation)
		(Knuth, programmation)
		...

avec les propriétés :

- *NIVEAU = année (1ère ou 2ème) concernée par l'enseignement d'une matière par un enseignant*
- *TYPE = est-ce un cours, un travail dirigé, ... ?*

....

. SEANCE (ENSEIGNANT, MATIERE, SALLE, GROUPE) [DUREE, TYPSALLE, ...]

Cette relation de degré 4 est l'ensemble des éléments constitutifs d'un emploi du temps : ensemble des séances de travail par un enseignant, sur une matière, dans une salle, à un groupe.

avec les propriétés :

- *DUREE du cours*
- *TYPSALLE : type d'un équipement spécialisé nécessaire (rétroprojecteur, magnétophone, ...)*
- ...

Attention : l'ensemble GROUPE n'est pas un ensemble de base au sens où nous l'avons défini (§ II.2.1.1) ; nous reviendrons sur sa description dans le paragraphe II.4.1.3.

Remarques concernant les arguments d'une relation

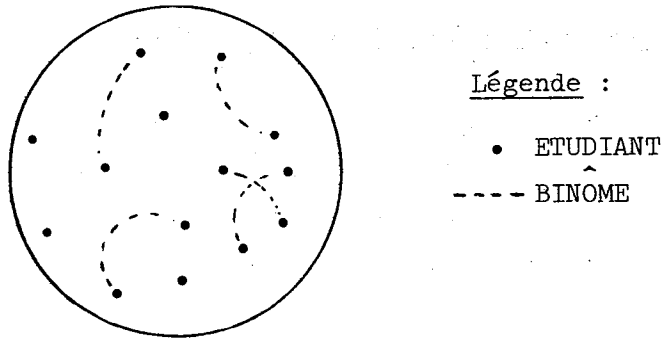
a) les arguments d'une relation peuvent être identiques

b) sur les mêmes ensembles-arguments, il est possible de construire des relations différentes

c) les relations sont elles-mêmes considérées comme de nouveaux ensembles sur lesquels on peut définir de nouvelles relations qui sont des relations de relations.

Exemple 2.14 :

- Pour regrouper les "binômes" de travaux pratiques de programmation, on peut décrire une relation binaire sur l'ensemble de base ETUDIANT (remarque a)
BINOME (ETUDIANT, ETUDIANT) [...].



- Avec les ensembles de base ENSEIGNANT, MATIERE, SALLE et GROUPE, on peut construire la relation SEANCE :
SEANCE (ENSEIGNANT, MATIERE, SALLE, GROUPE) [...]
si l'on s'intéresse à l'emploi du temps d'une semaine normale
ou la relation EXAMEN :
EXAMEN (ENSEIGNANT, MATIERE, SALLE, GROUPE) [...]
dans le cas du planning d'une semaine de devoirs surveillés (remarque b).

- A partir de la relation ENSEIGNEMENT :
ENSEIGNEMENT (ENSEIGNANT, MATIERE) [...]
on peut donner la relation COURS :
COURS (ENSEIGNEMENT, SALLE) [...]
qui elle-même permet d'obtenir la relation SEANCE2 :
SEANCE2 (COURS, GROUPE) [...]
(remarque c).

Remarque méthodologique :

Nous n'avons pas défini d'opération de projection d'une relation suivant un ou plusieurs de ses arguments. Aussi, la description de relations dont les arguments sont eux-mêmes des relations doit-elle être faite avec beaucoup de précautions . Dans le cas des exemples 2.13 et 2.14, rien ne permet d'affirmer que les deux relations

SEANCE (ENSEIGNANT, MATIERE, SALLE, GROUPE) [...]
et SEANCE2 (COURS, GROUPE) [...],

regroupent les mêmes éléments (cf. remarque b).

Nous pouvons simplement dire que :

- "la projection" (au sens mathématique) de la relation SEANCE2 suivant l'argument COURS correspond à l'ensemble des éléments de la relation COURS
- on ne construira que des relations sur lesquelles on veut décrire des transactions de gestion (cf. § II.2.3).

II.2.2.2 - Les propriétés-clés d'une relation

Comme les éléments d'un ensemble de base, tout élément d'une relation doit pouvoir être différencié des autres par la valeur discriminante d'une PROPRIETE qui sera appelée PROPRIETE-CLE de la relation. Pour toute relation, la définition d'au moins une PROPRIETE-CLE est obligatoire.

Comment définir une propriété-clé d'une relation ?

Il y a deux possibilités pour définir une propriété-clé d'une relation :

1 - l'identification d'un élément d'une relation peut, dans de nombreux cas, être faite par l'identification des divers éléments constituant le n-uplet : la clé est alors obtenue par la *concaténation des clés de ces éléments*.
Dans ce cas, la propriété-clé de la relation est une propriété composée dont les composantes sont les propriétés-clés des ensembles-arguments (cf. définition § II.2.1.4).

2 - Si l'on veut créer plusieurs liaisons distinctes sur les mêmes éléments, cette première solution ne peut pas convenir : la clé doit être alors *une propriété spécifique de la relation.*

Exemple 2.15 :

- *Un "binôme" peut être identifié simplement par les clés des deux étudiants si l'on admet qu'un étudiant ne peut collaborer qu'à un et à un seul binôme.*

  ETUDIANT [*NOM*, ...]

  BINOME (ETUDIANT, ETUDIANT) [*CLEBIN : (NOM || NOM)*, ...]

  La propriété-clé CLEBIN est une propriété composée des propriétés-clés des arguments.

- *La connaissance des clés d'un enseignant et d'une matière n'est pas toujours suffisante pour identifier un "enseignement" particulier si l'on suppose qu'un même enseignant peut enseigner une même matière dans différentes conditions : en cours, ou en travail dirigé, ...*

  ENSEIGNEMENT (ENSEIGNANT, MATIERE) [*CDENS, NIVEAU, TYPE*, ...]

  La propriété-clé CDENS est une propriété particulière de la relation.

II.2.2.3 - Définition d'invariants d'une relation : les fonctions de dépendance

Définition :

Pour toute relation N-aire, nous pouvons définir N FONCTIONS de DEPENDANCE pour décrire les contraintes sémantiques des diverses liaisons possibles entre les éléments des différents arguments.

Examinons d'abord le cas, fréquent, des relations binaires.

Fonctions de dépendance d'une relation binaire

Soient . une relation R (E,G),
. E' sous-ensemble quelconque de E,
. et G' sous-ensemble quelconque de G.

Alors on définit deux fonctions de dépendance f_E et f_G telles que :

. f_E : est la fonction de dépendance correspondant à l'argument E pour la relation R ;

et $f_E : G' \longrightarrow \mathcal{P}(E)$

$g \in G' \quad g \longrightarrow \mathcal{C} \in \mathcal{P}(E)$

$\forall e \in \mathcal{C}, \ \mathcal{C} = f_E(g) \iff (e,g) \in R$

. f_G : est la fonction de dépendance associée à l'argument G pour la relation R ;

$$f_G = E' \rightarrow \mathcal{P}(G)$$

$$e \in E' \qquad e \rightarrow \mathcal{G} \in \mathcal{P}(G)$$

$$\forall g \in \mathcal{G},\ \mathcal{G} = f_G(e) \iff (e,g) \in R$$

Pour chaque fonction de dépendance, nous définissons aussi le minimum et le maximum des cardinaux des sous-ensembles-images qu'elle décrit.

pour f_E : $MIN(f_E)$ = min de $\|\mathcal{C}\|$ $\forall g$

$MAX(f_E)$ = max de $\|\mathcal{C}\|$ $\forall g$

pour f_G : $MIN(f_G)$ = min de $\|\mathcal{G}\|$ $\forall e$

$MAX(f_G)$ = max de $\|\mathcal{G}\|$ $\forall e$

Nous donnons des noms différents à chaque fonction de dépendance (f_E, f_G) distincts du nom de la relation R ; ces deux noms, ainsi que les cardinaux MIN et MAX, apparaissent dans la notation utilisée à la suite de chaque argument.

R(E : (MIN,MAX) f_E, G : (MIN,MAX) f_G) [...]

Exemple 2.16 :

Soit la relation ENSEIGNEMENT (ENSEIGNANT, MATIERE) [...] ;
on a les deux fonctions de dépendance : "professeurs-de"
et "spécialités-de"

- *"professeurs-de" une MATIERE : c'est le nom d'un sous-ensemble de ENSEIGNANT dont les éléments sont liés à une MATIERE par la relation ENSEIGNEMENT*

  MIN ("professeurs-de") = 1, c'est dire qu'une MATIERE doit être enseignée par au moins un ENSEIGNANT

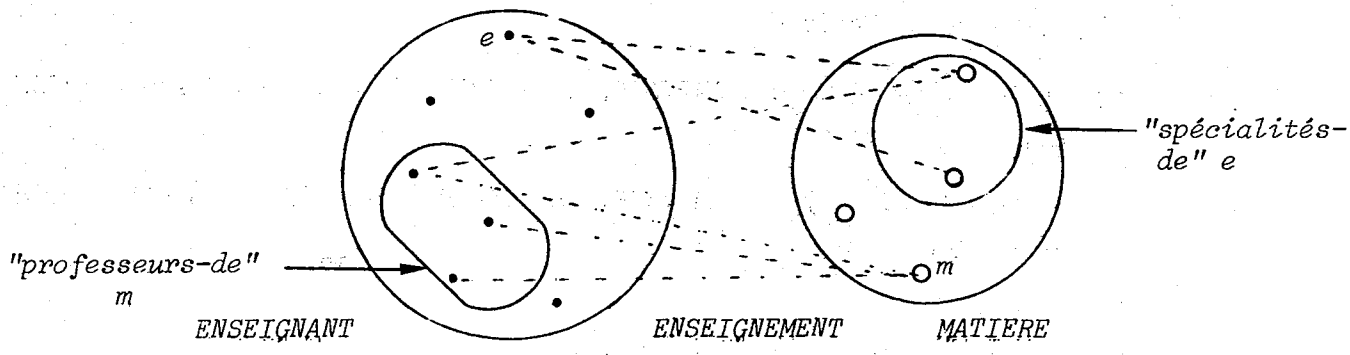
  MAX ("professeurs-de") = 10, 10 est le nombre maximum d'ENSEIGNANTS d'une MATIERE.

- *"spécialités-de" un ENSEIGNANT : c'est le nom d'un sous-ensemble de MATIERE dont les éléments sont liés à un ENSEIGNANT par la relation ENSEIGNEMENT.*

  MIN ("spécialités-de") = 1, signifie qu'un ENSEIGNANT doit être compétent dans au moins une MATIERE

  MAX ("spécialités-de") = 2, c'est dire qu'un ENSEIGNANT ne peut pas assurer l'enseignement de plus de deux MATIERES.

Soit ENSEIGNEMENT (ENSEIGNANT : (1,10) professeurs-de, MATIERE : (1,2) spécialités-de) [...]



Fonctions de dépendance d'une relation n-aire

Soit une relation $R(E_1, E_2, \ldots E_n)$.

Soit S un sous-ensemble quelconque de $E_1 \times E_2 \times \ldots \times E_{i-1} \times E_{i+1} \times \ldots \times E_n$ alors $s \in S : s = (e_1, e_2, \ldots e_{i-1}, e_{i+1}, \ldots e_n)$.

Pour chaque ensemble-argument E_i de la relation R, on définit une fonction de dépendance f_i par :

$$f_i : S \longrightarrow \mathcal{P}(E_i)$$
$$s \longrightarrow \mathcal{C}_i \in \mathcal{P}(E_i)$$

$$\forall e_i \in \mathcal{C}_i : (e_1, e_2, \ldots e_{i-1}, e_i, e_{i+1}, \ldots e_n) \in R \iff \mathcal{C}_i = f_i(s)$$

$$MIN(f_i) = \text{min de } \|\mathcal{C}_i\| \quad \forall s$$
$$MAX(f_i) = \text{max de } \|\mathcal{C}_i\| \quad \forall s$$

Exemple 2.17 :

Décrivons les fonctions de dépendance associées à la relation SEANCE (ENSEIGNANT, MATIERE, SALLE, GROUPE) ... :

arguments	*fonctions de dépendance*	*min*	*max*
ENSEIGNANT	*professeur*	*1*	*1*
MATIERE	*thèmes*	*1*	*2*
SALLE	*lieu*	*1*	*1*
GROUPE	*auditoires*	*1*	*4*

c'est dire que :

- pour une MATIERE, dans une SALLE, avec un GROUPE, il y a 1 et 1 seul

ENSEIGNANT "professeur" pour la SEANCE

- *pour un ENSEIGNANT, dans une SALLE, avec un GROUPE, il y a 1 ou 2 MATIERES "thèmes" de la SEANCE*
- *pour un ENSEIGNANT, une MATIERE, un GROUPE, il y a 1 et 1 seule SALLE "lieu" de la SEANCE*
- *pour un ENSEIGNANT, une MATIERE, une SALLE, il y a 1 à 4 GROUPES "auditeurs" pour la SEANCE.*

On la note :

SEANCE (ENSEIGNANT : (1,1) professeur,
MATIERE : (1,2) thèmes,
SALLE : (1,1) lieu,
GROUPE : (1,4) auditoires) [*DUREE, TYPSALLE, ...*]

II.2.2.4 - Remarques méthodologiques

a) Différence de nature entre ENSEMBLES de BASE, PROPRIETES et RELATIONS

En effet, on pourrait définir le NOM d'un ETUDIANT selon deux structures logiques des données distinctes :

STRUCTURE 1 : ensemble de base : "ETUDIANT"
Propriété "NOM" définie comme une propriété caractéristique de tous les éléments de l'ensemble ETUDIANT.

STRUCTURE 2 : ensemble de base : "ETUDIANT"
ensemble de base : "NOM"
relation binaire : "NOMMER" ayant pour arguments ETUDIANT et NOM, et associant à chaque étudiant un nom

NOMMER (ETUDIANT : de-nom, NOM : nom-de)

Ces deux structures ont une sémantique différente :

- dans la structure 1, la suppression de l'ensemble "ETUDIANT" implique celle de toutes ses propriétés, et en particulier de "NOM" car NOM est une propriété spécifique de ETUDIANT et de ETUDIANT seul.
- dans la structure 2, en revanche, la suppression de l'ensemble "ETUDIANT" implique la suppression de la relation "NOMMER" puisqu'un de ses arguments disparaît, mais l'ensemble "NOM"

posé en tant que tel, peut subsister, d'autant plus que NOM peut être argument d'autres relations.

Dans l'approche ensembliste de représentation des données que nous proposons, ce problème est résolu :

. Une PROPRIETE est une caractéristique associée exclusivement à un ensemble et à un seul. Si, après l'avoir créé, cet ensemble est supprimé, alors toutes les propriétés qui lui étaient associées sont supprimées implicitement.

. Nous ne conseillons la création d'une relation R entre deux ensembles E et F que s'il est possible de donner une réponse affirmative aux deux questions suivantes :

- E ou F sont-ils arguments de relations autres que R ?
- la suppression de la relation R entre E et F conserve-t-elle encore un sens à l'existence de F indépendamment de E ?

En général, la réponse négative à l'une de ces questions conduit à remplacer l'ensemble F et la relation R par la déclaration d'une propriété nouvelle à E.

b) Distinction entre les propriétés d'une relation et les propriétés de ses arguments

Soient deux ensembles E $[A_1, A_2, \ldots A_i, \ldots]$
F $[B_1, B_2, \ldots B_j, \ldots]$

et une relation R entre E et F

R (E : f_1(min1,max1), F : f_2(min2,max2)) $[P_1, P_2, \ldots, P_k, \ldots]$

Toute propriété P_k de R peut induire une propriété A'_k de E :

$A'_k(e) = P_k((e,f))$, $\forall e \in E$ si $\exists f \in F$ et $(e,f) \in R$

Il serait alors faux de considérer les propriétés de l'ensemble E (ou F) comme étant aussi celles de la relation R et réciproquement, car :

- si min1 = 0 (fonction de dépendance f_1 de la relation R), alors A'_k n'est pas partout définie sur E

- si max2 > 1 ⇒ ∃f et f' : R(e,f) et R(e,f'), alors A'_k aura plusieurs valeurs pour un même élément e de E.

Dans ces deux cas A'_k n'est pas une propriété de E au sens où nous l'avons défini en posant les conventions § II.2.1.2 :

1 - Une propriété doit être propriété de tous les éléments d'un ensemble.

2 - Une propriété doit avoir une valeur et une seule pour chaque élément d'un ensemble.

3 - Une propriété d'une relation peut changer de valeur pour chaque n-uplet.

Ces contraintes sémantiques sur la définition d'une propriété permettent d'éviter la description, au niveau d'un ensemble, d'une propriété qui est, en fait, celle d'une relation dont l'ensemble est argument.

c) Relations n-aires et décompositions de relations n-aires

Le modèle proposé laisse toute liberté à l'utilisateur pour définir des relations binaires ou n-aires entre ensembles ou (et) relations. Ces diverses possibilités conduisent à plusieurs remarques.

Remarque 1 : Si lors de la création d'une relation n-aire R (E_1, E_2, ... E_n) il s'avère impossible de décrire une des fonctions de dépendance f_i, alors il faut envisager de définir, de préférence, la relation (n-1) aire : R'(E_1, ... E_{i-1}, E_{i+1}, ... E_n) qui n'a pas E_i comme argument.

Remarque 2 : Les n fonctions de dépendance fi d'une relation n-aire R(E_1, E_2, ... E_n) sont des caractéristiques exclusives de chaque argument par rapport aux (n-1) autres. Cette restriction de description n'interdit pas la formulation de contraintes relationnelles supplémentaires au niveau du "texte de définition" de la relation :

- indiquer une dépendance d'un argument par rapport à (n-q) autres avec q > 1
- ou une dépendance de p arguments par rapport à (n-q) autres avec p ≤ q.

Si l'utilisateur estime ces contraintes suffisamment fortes, il décidera lui-même de l'opportunité des créations éventuelles de nouvelles relations :

relation (n-q+1)-aire pour le premier cas, et dans le second cas relations R' p-aire, R"(n-q)-aire et relation binaire R"'(R',R")
(cf. Remarques § II.2.2.1).

II.2.2.5 - Conclusions et représentations graphiques provisoires

Nous avons énoncé deux définitions :

- les relations n-aires,
- les fonctions de dépendance associées aux arguments d'une relation ;

et décrit les caractéristiques principales des relations.

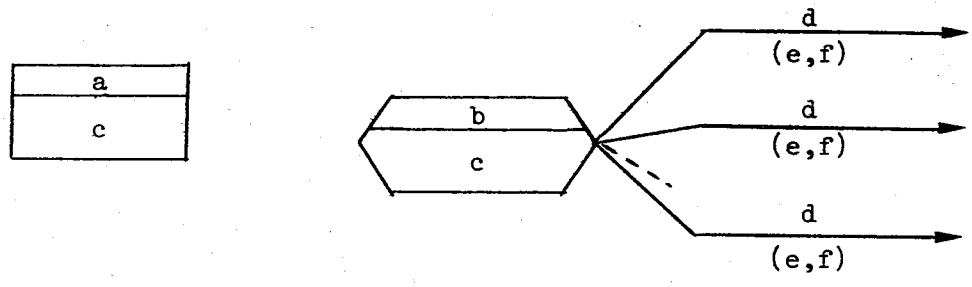
Pour chaque RELATION, il faut préciser :
- des caractéristiques fondamentales :
  . un CODE,
  . une ou plusieurs PROPRIETES-CLES,
  . des PROPRIETES,
  . ses ARGUMENTS en précisant pour chacun d'eux le NOM, le MIN et le MAX de la FONCTION de DEPENDANCE associée.

- des caractéristiques qualitatives :
  . un LIBELLE,
  . une DEFINITION,
  . une TAILLE.

Note :
Les caractéristiques CODE, PROPRIETES, CLES, LIBELLE, DEFINITION et TAILLE ont les mêmes significations que pour les ensembles de base.

Représentation graphique d'une structure de données

Pour faciliter des besoins de synthèse, après de nombreuses descriptions d'ensembles de base ou de relations concernant une structure de données, nous proposons un symbolisme pour représenter graphiquement cette structure.



a : CODE d'un ensemble de base

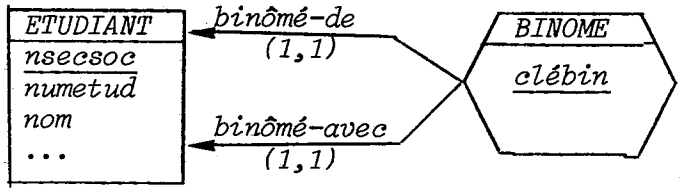
b : CODE d'une relation

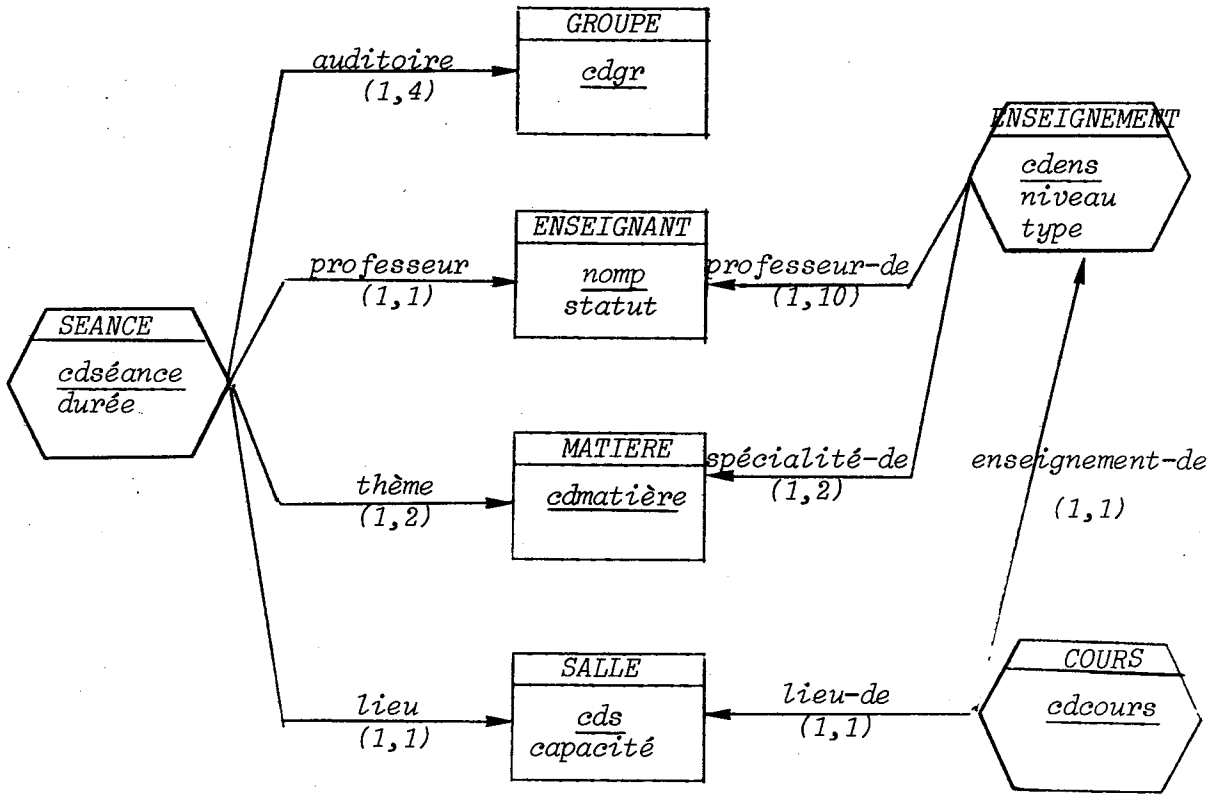
c : CODES de propriétés-clés + éventuellement quelques codes d'autres propriétés

d : CODE d'une fonction de dépendance

(e,f) : MIN et MAX d'une fonction de dépendance.

Exemple 2.18 :





II.2.3 - Les transactions et leurs schémas logiques

II.2.3.1 - Notion de transaction

Définition :

Une transaction est une OPERATION AUTOMATISEE, non interruptible, d'ECHANGE d'INFORMATIONS entre le système informatique et l'utilisateur, en ENTREE, en SORTIE ou en ENTREE-SORTIE.

Une transaction est "non interruptible", car elle doit se dérouler complètement si elle est commencée et elle est acceptée ou rejetée dans son ensemble au cours de sa validation.

Dans d'autres domaines, diverses dénominations sont utilisées : message en technique de transmission, requête pour un système temps réel, ...

Si nous nous intéressons aux aspects externes (visuels) de ces opérations, nous pouvons dire qu' :

- une transaction en entrée peut être : un document perforé, un document dactylocodé sur support magnétisé, un format d'écran, une procédure conversationnelle, ...
- une transaction de sortie peut aussi s'afficher sur de nombreux périphériques : document papier (imprimante, machine à écrire), écran, microfilm, ...
- une transaction d'entrée et de sortie utilisera divers supports, comme une machine à écrire et un écran par exemple.

Précisons par des exemples ces opérations dans les trois cas fréquents que sont :

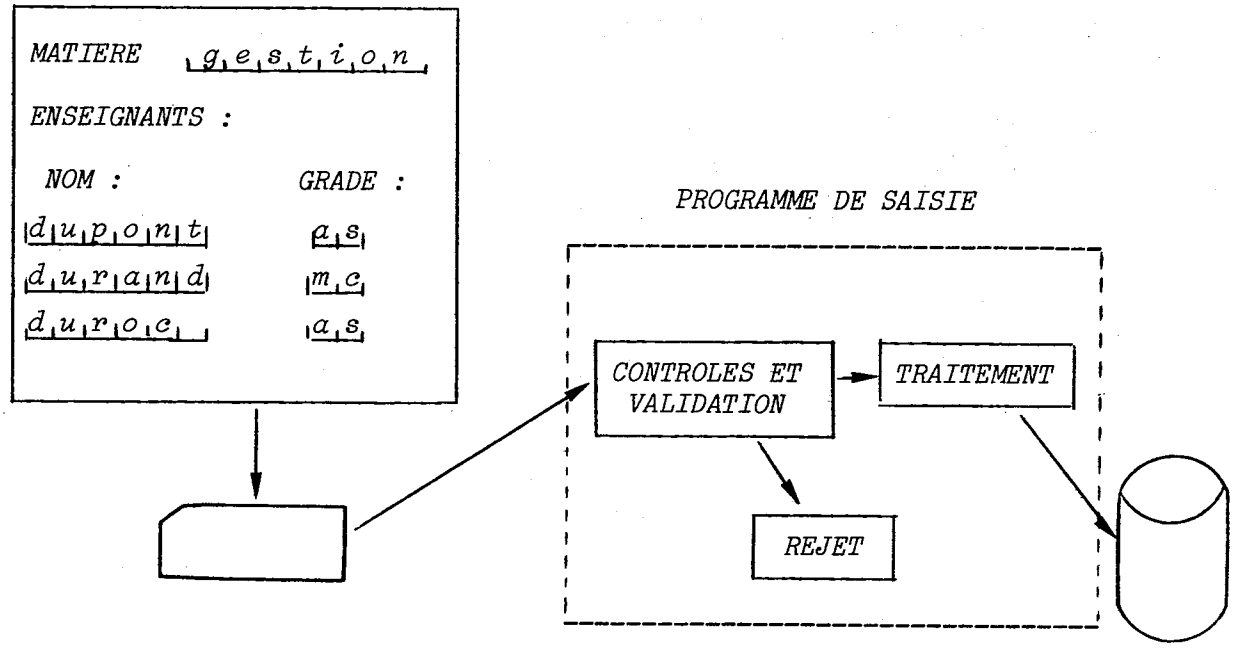
. l'entrée de données en traitement par lots ;
. l'entrée conversationnelle de données ;
. la sortie de données en traitement par lots ou en conversationnel.

La saisie en traitement par lots : le document d'entrée

L'utilisateur remplit des documents zonés (bordereaux de perforation), qui sont ensuite perforés et lus par un programme de saisie assurant les contrôles de validation, le traitement ou le rejet éventuel.

Ce document zoné est souvent en partie préimprimé (titre du document, nom de chaque zone, ...).

Exemple 2.19 :



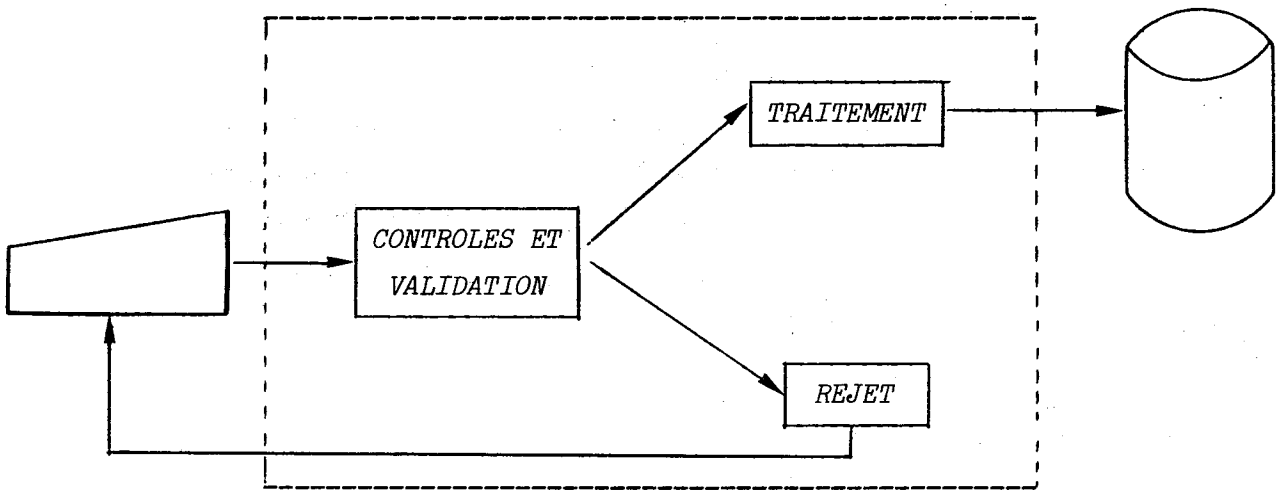
La saisie conversationnelle : le dialogue

Devant son terminal, l'utilisateur répond à des questions.

Exemple 2.20 :

MATIERE ? gestion
VOULEZ-VOUS LUI AFFECTER ENCORE UN ENSEIGNANT ? oui
NOM ? dupont
GRADE ? as
VOULEZ-VOUS LUI AFFECTER ENCORE UN ENSEIGNANT ? oui
NOM ? durand
GRADE ? mc
VOULEZ-VOUS LUI AFFECTER ENCORE UN ENSEIGNANT ? oui
NOM ? duroc
GRADE ? as
VOULEZ-VOUS LUI AFFECTER ENCORE UN ENSEIGNANT ? non
VOUS VENEZ D'AFFECTER 3 ENSEIGNANTS EN GESTION
MATIERE ? anglais
...

En cas de réponse erronée, la question lui est posée à nouveau.

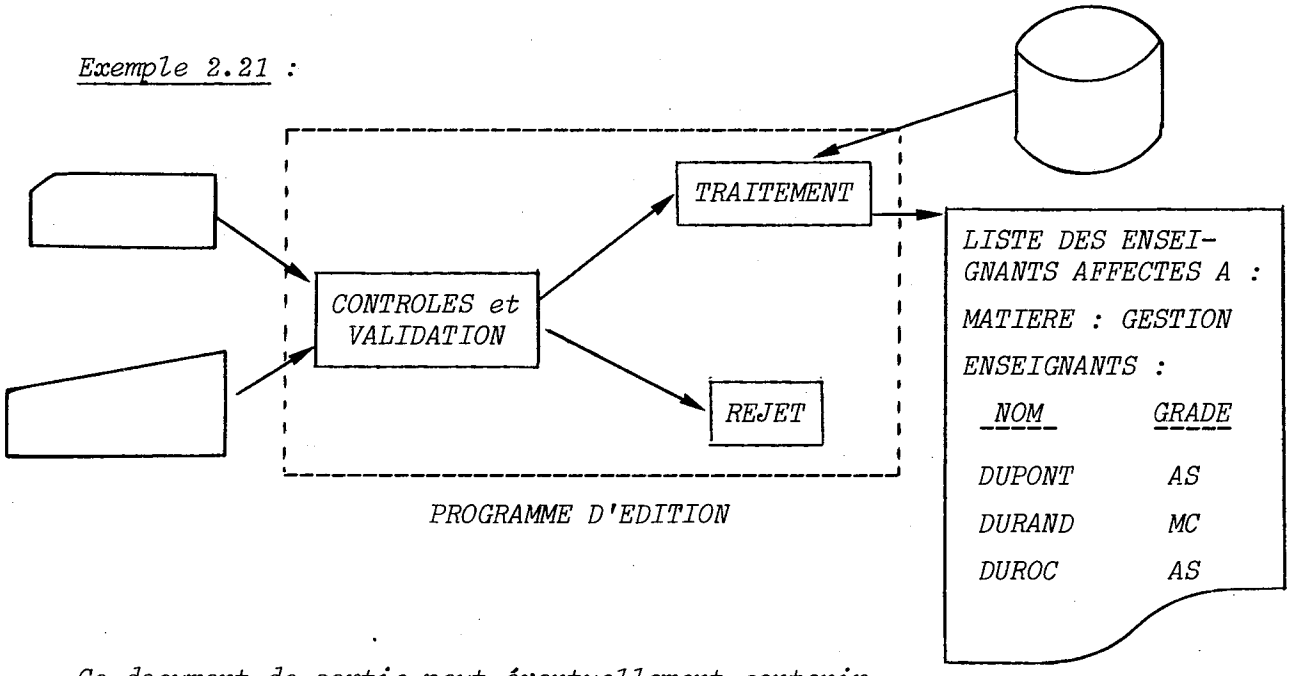


PROGRAMME DE SAISIE

L'édition batch ou conversationnelle : l'imprimé

L'utilisateur demande l'exécution d'un programme d'édition ; sa demande peut être formulée en batch ou en conversationnel et les résultats sont sortis sur l'imprimante, le terminal ou l'écran.

Exemple 2.21 :



PROGRAMME D'EDITION

Ce document de sortie peut éventuellement contenir des éléments préimprimés (titres, ...).

Remarques :

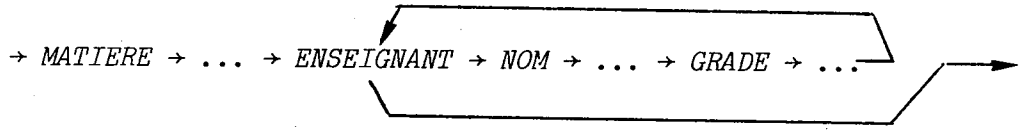
a) Les documents, dialogues et imprimés précédents sont tous composés de deux types d'informations :

- des VALEURS (dupont, durand, mc, ...),
- des SIGNIFIANTS de valeurs (NOM, GRADE, ...) ; mots-clés ou zones réservés sur un document ou dans un message qui peuvent être associés de façon biunivoque à des caractéristiques d'ensembles.

Les VALEURS et les SIGNIFIANTS se succèdent suivant un ordre logique qui peut être représenté graphiquement.

Exemple 2.22 :

Reprenons l'exemple précédent :



Les pointillés (...) sont à remplacer par les valeurs qui seront lues ou écrites effectivement lors de l'exécution de l'opération d'entrée ou de sortie.

b) Un tel graphe décrit l'algorithme de l'opération envisagée, il constitue la définition d'une "grille" (ou "guide") où se trouvent répertoriés tous les cas possibles d'échanges prévus.

L'activation d'une opération ainsi définie peut être représentée par une phrase construite conformément au graphe.

Exemple 2.23 :

Soit une phrase écrite suivant le graphe de l'exemple 2.22 :

MATIERE gestion ENSEIGNANT NOM dupont GRADE as ENSEIGNANT ...

II.2.3.2 - Description d'une transaction

CODE, LIBELLE, DEFINITION et SENS d'une transaction

Comme les ensembles et les propriétés (cf. § II.2.1.1 et II.2.1.4), les transactions sont caractérisées par :

- un CODE d'identification discriminant,
- un LIBELLE explicatif,
- une DEFINITION complète.

Mais il faut indiquer aussi :

- son SENS pour préciser le sens de l'échange qui peut être une ENTREE, SORTIE ou ENTREE-SORTIE de données.

Exemple 2.24 :

Une première transaction :

. *CODE : TECREPROF*
. *LIBELLE : enregistrement d'un nouvel enseignant*
. *DEFINITION : quand un nouvel enseignant est nommé dans un département ou qu'un nouveau vacataire doit assurer des heures d'enseignement, il faut enregistrer ses caractéristiques pour mettre à jour la liste des enseignants du département pour l'année en cours.*
. *SENS : entrée.*

Une deuxième transaction :

. *CODE : TEMODENS*
. *LIBELLE : description de nouveaux enseignements pour un enseignant*
. *DEFINITION : chaque enseignant peut assurer un ou plusieurs enseignements d'une certaine matière dans l'une des deux années d'études. Cette transaction est utilisée pour décrire de nouveaux enseignements pour un enseignant donné.*
. *SENS : entrée.*

Une troisième transaction :

. *CODE : TSLISTPROF*
. *LIBELLE : édition de la liste des enseignants de l'année en cours*
. *DEFINITION : éditer la liste des enseignants en indiquant pour chaque enseignant son numéro de sécurité sociale, son nom, son prénom et les enseignements qu'il assure.*
. *SENS : sortie.*

Un schéma de transaction

Le schéma d'une transaction indique l'ordre logique
- d'introduction par l'utilisateur (ENTREE),
- de présentation à l'utilisateur (SORTIE),

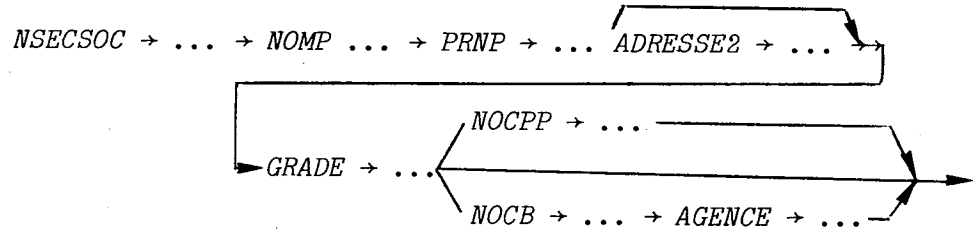
des données manipulées par une transaction.

Nous le représentons par un GRAPHE.

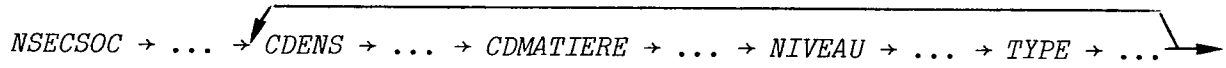
Exemple 2.25 :

Soient les ensembles ENSEIGNANT [NSECSOC, NOMP, PRNP, ADRESSE2, GRADE, ...]
MATIERE [CDMATIERE, ...]
et la relation ENSEIGNEMENT (ENSEIGNANT, MATIERE) [CDENS, NIVEAU, TYPE, ...]

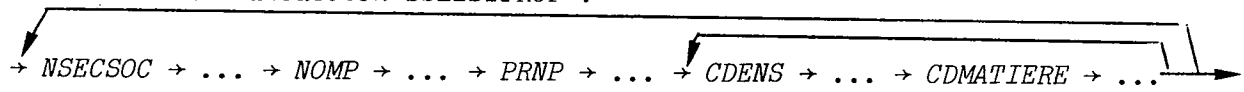
SCHEMA de la transaction TECREPROF :



SCHEMA de la transaction TEMODENS :



SCHEMA de la transaction TSLISTPROF :



Une réalisation de transaction

Une réalisation d'une transaction est une occurrence particulière d'une transaction ; c'est-à-dire une phrase décrivant toutes les données réelles à échanger, et respectant l'ordre précisé par le schéma de cette transaction.

Exemple 2.26 :

Soient deux réalisations de la transaction TECREPROF :
r1 = NSECSOC 1490538130030 NOMP dubois PRNP jean ADRESSE2 1 rue des lilas - 38000 grenoble

GRADE as NOCCP 12125 R

r2 = NSECSOC 2470338196176 NOMP durand PRNP marie GRADE ma

1 réalisation de la transaction TEMODENS :

r3 = NSECSOC 1490538130030 CDENS e42 CDMATIERE analyse NIVEAU 2A TYPE td
CDENS e43 CDMATIERE cobol NIVEAU 1A TYPE tp

1 réalisation de la transaction TSLISTPROF :

r4 = NSECSOC : 1490538130030
NOMP : DUBOIS PRNP : JEAN CDENS : E42 CDMATIERE : ANALYSE
CDENS : E43 CDMATIERE : COBOL
NSECSOC : 2470338196176
NOMP : DURAND PRNP : MARIE CDENS : E17 CDMATIERE : FORTRAN
NSECSOC : 1520669045215
NOMP : DUROC PRNP : PAUL CDENS : E04 CDMATIERE : GESTION
...

Note :
Dans la suite de l'exposé, nous ne décrivons que des transactions d'ENTREE ou des transactions de SORTIE de données ; le cas des transactions d'ENTREE-SORTIE des données peut facilement en être déduit, mais il compliquerait inutilement la présentation des concepts fondamentaux du modèle, en particulier les schémas des transactions : il faudrait alors indiquer au niveau de chaque donnée d'un schéma si elle doit être lue ou écrite.

II.2.3.3 - Règles de construction d'un schéma de transaction

Champ d'un schéma d'une transaction

Un schéma de transaction est un graphe composé de zones élémentaires appelées CHAMPS.

Un champ est constitué :

- d'un signifiant de zone,
- d'une valeur à lire ou à écrire.

Exemple 2.27 :

NOMP → ...
ADRESSE2 → ...
CDMATIERE → ...
NIVEAU → ... *sont des champs des schémas des transactions TECREPROF et TEMODENS*

Convention :

Les SIGNIFIANTS de zones doivent être des CODES de PROPRIETES du modèle des données, et chaque VALEUR doit appartenir à l'ESPACE de VALEURS de la propriété correspondante.

Un champ est alors entièrement spécifié par la connaissance du CODE de la propriété concernée.

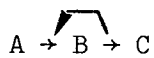
Exemple 2.28 :

- *Dans la transaction TECREPROF : NSECSOC, NOMP, PRNP, ADRESSE2, ... sont les codes de propriétés de l'ensemble ENSEIGNANT.*
- *Dans la transaction TEMODENS,*
  - *CDENS, NIVEAU, TYPE sont les codes de propriétés de la relation ENSEIGNEMENT,*
  - *CDMATIERE est le code d'une propriété de l'ensemble MATIERE.*

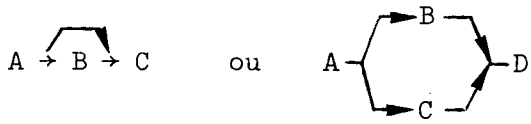
CHEMINS d'un schéma d'une transaction

Nous appelons CHEMIN une succession de champs ; un chemin peut être :

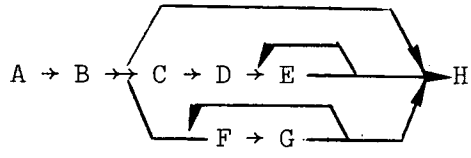
- simple A → B → C
- itératif



- multiple



- une combinaison entre les 3 possibilités :



Un schéma de transaction est un chemin qui est en général une combinaison de chemins simples, itératifs et multiples.

Remarque :
Le schéma d'une transaction peut être lui-même itératif, comme par exemple celui de la transaction de sortie TSLISTPROF (exemple 2.25).

Ensemble de référence d'une transaction

Pour définir le "domaine d'application" d'une transaction, nous convenons qu'une réalisation d'une transaction T ne peut porter que sur des échanges de valeurs :

- de PROPRIETES d'éléments d'un ensemble E (ensemble de base ou relation) du modèle des données,
- éventuellement de PROPRIETES d'éléments d'une relation R, dont E est argument, et/ou de PROPRIETES-CLES d'éléments d'ensembles-arguments de R.

E est appelé ENSEMBLE de REFERENCE de la transaction T.

Exemple 2.29 :

Pour la transaction TECREPROF, toutes les propriétés concernées appartiennent à l'ensemble ENSEIGNANT qui est son ensemble de référence.

Dans le cas de la transaction TEMODENS, l'ensemble de référence est ENSEIGNANT, mais on donne aussi :

- *des propriétés de la relation ENSEIGNEMENT (ENSEIGNANT, MATIERE), comme CDENS, NIVEAU, TYPE*
- *une propriété-clé (CDMATIERE) de l'ensemble MATIERE argument de cette relation ENSEIGNEMENT.*

Règles restrictives pour la construction des schémas de transaction

Règle 1 : Soit une transaction T, d'ensemble de référence E, et une relation R(E, ... E_i, ... E_n).

Tous les CHAMPS du schéma de la transaction T peuvent être décrits par :

- des CODES de PROPRIETES de l'ensemble de référence E,
- des CODES de PROPRIETES de la relation R(E,...),
- ou des CODES de PROPRIETES-CLES des arguments E_i de la relation R.

Exemple 2.30 :

. *Dans le schéma de la transaction TECREPROF, tous les CHAMPS (NSECSOC, NOMP, PRNP, ADRESSE2, GRADE, NOCCP, NOCB, AGENCE) sont décrits par des codes de propriétés de l'ensemble de référence ENSEIGNANT.*

. *Pour le schéma de TEMODENS :*

- *NSECSOC : propriété de l'ensemble de référence ENSEIGNANT*
- *CDENS, NIVEAU, TYPE : propriétés de la relation ENSEIGNEMENT (ENSEIGNANT, MATIERE)*
- *CDMATIERE : propriété-clé de l'ensemble MATIERE, argument de la relation ENSEIGNEMENT.*

Règle 2 : La description d'un SCHEMA de TRANSACTION commence toujours par un CHEMIN SIMPLE composé de CHAMPS correspondant à des CODES de propriétés de l'ENSEMBLE de REFERENCE et dont le premier est une PROPRIETE-CLE.

Ainsi, une transaction débute toujours par l'identification d'un élément de l'ensemble de référence.

Exemple 2.31 :

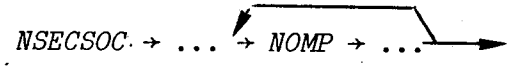
NSECSOC → ... → NOMP → ... → PRNP → ... →
est un chemin simple de la transaction TECREPROF.

Règle 3 : Dans un schéma de transaction, un chemin itératif ne peut pas être composé de champs correspondant à des propriétés de l'ensemble de référence de la transaction ; un tel chemin est construit avec des propriétés d'une relation et/ou de ses arguments.

Cette règle est conforme à la condition d'unicité de la valeur d'une propriété d'un élément d'un ensemble (cf. § II.2.1.2).

Exemple 2.32 :

Dans le schéma d'une transaction T avec l'ensemble de référence ENSEIGNANT, on ne peut pas avoir le chemin suivant :

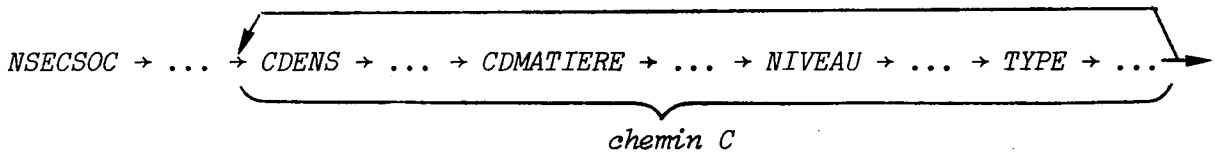


La personne aurait plusieurs NOMS, et il faut plusieurs propriétés représentatives de ceux-ci.

Remarque :
Un chemin d'un schéma de transaction ne peut être itératif que si les cardinalités des fonctions de dépendances concernées l'autorisent.

Exemple 2.33 :

Soient le schéma :



et la relation :
ENSEIGNEMENT (ENSEIGNANT : (1,10) professeur-de, MATIERE : (1,2) spécialités-de) [...]
alors le chemin C peut être itératif, et parcouru deux fois, car la fonction de dépendance "spécialités-de" (un ENSEIGNANT) admet la valeur 2 comme cardinal maximum.

Soient des réalisations de cette transaction :

r1 : NSECSOC 1490538130030 CDENS e42 CDMATIERE analyse NIVEAU 2A TYPE td
CDENS e43 CDMATIERE cobol NIVEAU 1A TYPE tp

r2 : NSECSOC 2470338196176 CDENS e57 CDMATIERE gestion NIVEAU 1A TYPE td

r3 : NSECSOC 1510769237024 CDENS e17 CDMATIERE fortran NIVEAU 1A TYPE tp
CDENS e22 CDMATIERE cobol NIVEAU 1A TYPE tp
CDENS e23 CDMATIERE anglais NIVEAU 2A TYPE tp

Les réalisations r1 et r2 sont correctes, alors que r3 est erronée : un enseignant ne peut enseigner que dans deux matières au plus.

II.2.3.4 - Conclusion

Un transaction est définie par :

- son CODE,
- son LIBELLE,
- sa DEFINITION,
- son ENSEMBLE de REFERENCE,
- son SENS (entrée, sortie, entrée-sortie),
- son SCHEMA de transaction.

Le schéma d'une transaction décrit logiquement l'opération d'échange de données envisagée. Des règles assurent des constructions de schémas

compatibles avec le modèle des données.

II.3 - INTRODUCTION DE L'ASPECT DYNAMIQUE DU MODELE

II.3.1 - Bilan provisoire sur les caractéristiques statiques

Nous avons décrit la plupart des caractéristiques *STATIQUES* d'une application, c'est-à-dire :

- d'une part les DONNEES :
  - . leur IDENTIFICATION (code d'une propriété, d'un ensemble ou d'une relation) ;
  - . leur SIGNIFICATION (libellé et définition) ;
  - . leurs CONTRAINTES d'intégrité (valeurs des espaces de valeurs d'une propriété, taille d'un ensemble ou d'une relation, fonctions de dépendance d'une relation vis-à-vis de ses arguments) ;
- d'autre part les TRANSACTIONS :
  - . leur IDENTIFICATION et leur SIGNIFICATION (code, libellé, définition, et sens d'une transaction) ;
  - . leur CONTENU (ensemble de référence, champs et schéma d'une transaction).

Les DONNEES et les TRANSACTIONS sont étroitement liées dans ce modèle par les notions d'ensembles de référence et de champs de schémas de transactions.

Mais un système d'informations fonctionne *DYNAMIQUEMENT* car les opérations envisagées à un moment donné tiennent compte des opérations exécutées auparavant.

Ainsi, la réalisation d'une transaction concernant un élément de son ensemble de référence peut :
- CREER cet élément s'il n'existe pas encore,
- MODIFIER ou SUPPRIMER cet élément s'il existe déjà,

dans les fichiers ou bases de données où sont stockées les valeurs de ses propriétés introduites par cette réalisation. Nous devons énoncer avec précision la sémantique de ces opérations de base.

II.3.2 - Opérations élémentaires définies sur les éléments d'un ensemble de base ou d'une relation

II.3.2.1 - Déclarations d'un élément d'un ensemble de base

Définition :

DECLARER un nouvel élément c'est fournir la liste minimum des valeurs des propriétés de cet élément pour l'identifier et le caractériser dans le système d'informations.

La déclaration d'un élément contient TOUJOURS la valeur d'une PROPRIETE-CLE de l'ensemble, afin de permettre son identification sans ambiguïté.

II.3.2.2 - Sémantique de déclaration d'un ensemble de base

Convention :

On appelle SEMANTIQUE de DECLARATION (en abrégé S.D.) d'un ensemble de base la LISTE (non vide) des PROPRIETES de cet ensemble dont il faut fournir OBLIGATOIREMENT les valeurs pour qu'un élément soit considéré comme EXISTANT dans l'application informatique.

Il y a toujours une PROPRIETE-CLE de l'ensemble qui appartient à cette liste.

On devrait dire, plus précisément, sémantique de déclaration de tout élément d'un ensemble de base.

Notation :

Dans les notations des ensembles et des éléments, nous soulignerons d'un trait continu les clés et d'un trait discontinu les propriétés de la sémantique de déclaration.

Exemple 3.1 :

Pour assurer l'existence d'un ETUDIANT, nous pouvons décider qu'il faut connaître son NOM, son PRENOM et son NUMERO qui constituent la sémantique de sa déclaration.

Nous notons l'ensemble de base ETUDIANT :

ETUDIANT [NOM, PRN, ADRESSE, NUMETUD, INSEE, ...]

Pour les ensembles SALLE et MATIERE :

SALLE [*CDS*, *CDBAT*, *ETAGE*, *CAPACITE*, ...]

MATIERE [*CDMATIERE*, *RESUME*, ...]

Toute transaction d'une application informatique, qui assure l'ajout d'un élément d'un ensemble, doit réaliser la saisie des informations relatives à cette sémantique de déclaration et doit exécuter aussi les contrôles nécessaires de vérification d'existence de cet élément dans les fichiers et bases de données où sont catalogués les éléments de l'ensemble.

Exemple 3.2 :

Soient des éléments de l'ensemble ETUDIANT :

. *(dubois, jean, 1 rue des fleurs à Grenoble, 750754, U, ...)*

La S.D. de cet étudiant est complète, il peut donc exister dans le système.

. *(dupont, U, U, 730168, 1550538130030, ...)*

Cet étudiant a une S.D. incomplète : son PRENOM doit avoir une valeur définie pour assurer son existence. Cet étudiant n'est pas encore déclaré dans le système.

II.3.2.3 - Déclaration d'un élément d'une relation

Nous avons déjà affirmé que (cf. § II.2.2) les relations doivent être considérées comme des ensembles ; ainsi leurs éléments (liaisons entre d'autres éléments) peuvent être créés, modifiés ou supprimés dans l'application informatique.

Pour que l'analogie soit complète, on définit aussi l'opération de DECLARATION d'un élément d'une relation.

Définition :

DECLARER un nouvel élément d'une relation, c'est fournir la liste minimum de valeurs de propriétés pour :

- identifier et caractériser cet élément dans le système d'informations,
- identifier les éléments des ensembles-arguments, ainsi mis en relation.

La déclaration d'un élément d'une relation contient toujours la valeur d'une propriété-clé l'identifiant sans ambiguïté.

Soit une relation R (E,F,...,G) [...] , la déclaration d'un élément r ∈ R, r = (e,f,...,g), peut prendre des formes très diverses :

- e, f, ... et g existent déjà dans le système d'informations ; "déclarer r" c'est identifier e, f, ... et g par leurs valeurs clés et fournir des valeurs de propriétés de r.
- e, f et/ou g n'existent encore pas dans le système ; "déclarer r" consiste alors à déclarer les éléments arguments qui n'existent pas, à identifier ceux qui existent et à fournir des valeurs de r.

Exemple 3.3 :

Soit la relation ENSEIGNEMENT (ENSEIGNANT, MATIERE) [CDENS, TYPE, NIVEAU, ...]
ENSEIGNANT [NOMP, ...]
MATIERE [CDMATIERE, RESUME, ...]

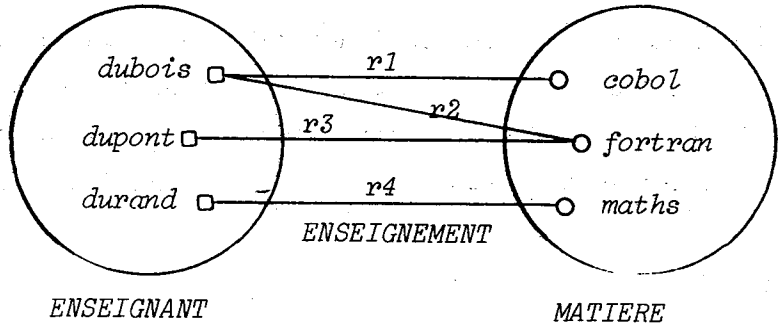
Supposons qu'un élément de ENSEIGNANT et deux éléments de MATIERE sont déjà déclarés

. *Pour déclarer les éléments r1 et r2 de la relation ENSEIGNEMENT*
r1 = (dubois, cobol)
r2 = (dubois, fortran)
nous devons :
- *identifier ces éléments par une valeur clé (CDENS)*
- *"repérer" l'enseignant "dubois" et les matières "fortran " et "cobol",*
- *caractériser les éléments r1 et r2 par éventuellement les valeurs de TYPE, NIVEAU, ...,*

. *Pour déclarer l'élément r3 (dupont, fortran), il faut :*
- *déclarer l'élément "dupont" dans l'ensemble ENSEIGNANT,*
- *"repérer" la matière "fortran" et l'enseignant "dupont",*
- *identifier et caractériser r3 par CDENS, TYPE, NIVEAU, ...*

. *Pour déclarer l'élément r4 (durand, maths), il faut :*

- *déclarer l'élément "durand" dans l'ensemble ENSEIGNANT,*
- *déclarer l'élément "maths" dans l'ensemble MATIERE,*
- *"repérer" l'enseignant "durand" et la matière "maths",*
- *identifier et caractériser r4.*



Convention :

Nous introduisons une restriction importante :
on ne peut DECLARER un élément d'une relation, c'est-à-dire un n-uplet, que si les éléments du n-uplet ont déjà été déclarés dans les ensembles arguments.

Autrement dit, l'opération de déclaration d'un élément d'une relation ne peut être utilisée que pour déclarer cet élément et non pour déclarer des éléments des ensembles-arguments.

Exemple 3.4 :

Dans l'exemple précédent, l'opération de déclaration de l'élément r3 doit être précédée (mais ne contient pas) de l'opération de déclaration de l'élément "dupont".

Pour r4, on fera successivement les trois opérations indépendantes :

- *déclaration de l'élément "durand" ;*
- *déclaration de l'élément "maths" ;*
- *déclaration de l'élément r4 (durand, maths).*

II.3.2.4 - Sémantique de déclaration d'une relation

La SEMANTIQUE de DECLARATION (des éléments) d'une RELATION est la liste minimum des PROPRIETES dont il faut fournir OBLIGATOIREMENT les valeurs pour qu'un élément soit DECLARE dans le système d'informations.

Cette liste contient TOUJOURS :

- une PROPRIETE-CLE de la relation (pour identifier un élément de la relation),
- une PROPRIETE-CLE relative à chaque argument de la relation (pour repérer les éléments à mettre en relation),
- éventuellement d'autres PROPRIETES de la relation (pour mieux caractériser l'élément de la relation).

Pour la sémantique de déclaration d'une relation, nous utilisons alors les mêmes conventions de notation que pour les ensembles de base, à ceci près que les propriétés-clés des arguments n'apparaissent pas dans cette notation car elles sont supposées implicites, compte tenu de la remarque précédente.

Exemple 3.5 :

Soient les relations :

BINOME (ETUDIANT, ETUDIANT) [*CLEBIN*, ...]

ENSEIGNEMENT (ENSEIGNANT, MATIERE) [*CDENS*, *TYPE*, *NIVEAU*, ...]

SEANCE (ENSEIGNANT, MATIERE, SALLE, GROUPE) [*CDSEANCE*, *DUREE*, *TYPSALLE*, ...]

On a les sémantiques de déclaration :

pour BINOME :
- *CLEBIN (clé de la relation),*
- *NUMETUD (clé du premier étudiant),*
- *NUMETUD (clé du deuxième étudiant).*

ENSEIGNEMENT :
- *CDENS (clé de la relation),*
- *NOMP (clé de l'enseignant),*
- *CDMATIERE (clé de la matière),*
- *TYPE et NIVEAU (propriétés de la relation).*

SEANCE :
- *CDSEANCE (clé de la relation),*
- *NOMP (clé de l'enseignant),*
- *CDMATIERE (clé de la matière),*
- *CDS (clé de la salle),*
- *CDGRP (clé du groupe),*
- *DUREE (propriété de la relation).*

II.3.2.5 - DECLARATION, DESCRIPTION, CREATION, CITATION, MODIFICATION et SUPPRESSION d'un élément d'un ensemble

Ces opérations élémentaires sont définies pour tout élément d'un ensemble, qu'il soit ensemble de base ou relation.

DECLARATION d'un élément

C'est l'opération qui réalise l'AFFECTATION D'UNE VALEUR définie pour chaque propriété appartenant à la SEMANTIQUE DE DECLARATION de l'ensemble auquel appartient le nouvel élément.
(cf. conventions § II.3.2.1 et § II.3.2.4).

DESCRIPTION d'un élément

C'est AJOUTER à un élément DEJA DECLARE des valeurs de certaines ou de toutes les propriétés qui ne font pas partie de sa sémantique de déclaration.

CREATION d'un élément

C'est assurer SUCCESSIVEMENT pour un élément sa DECLARATION et sa DESCRIPTION.

CITATION d'un élément

C'est l'IDENTIFICATION par sa valeur pour une propriété-clé, d'un élément DEJA DECLARE et/ou DECRIT et/ou CREE.

MODIFICATION d'un élément

C'est CHANGER les valeurs de certaines propriétés d'un élément déjà DECLARE et/ou DECRIT et/ou CREE.
Mais il ne peut pas y avoir de modification de la valeur d'une propriété-clé.

SUPPRESSION d'un élément

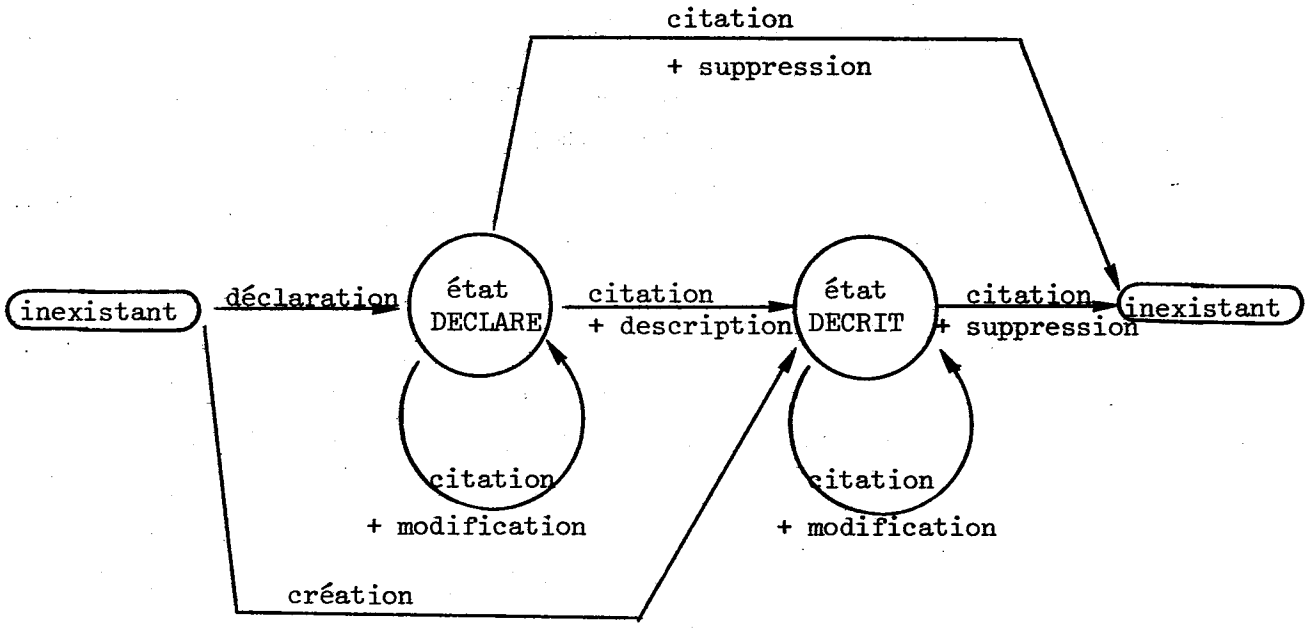
C'est EFFACER TOUTES les valeurs de toutes les propriétés d'UN élément déjà DECLARE et/ou DECRIT et/ou CREE et/ou MODIFIE.

Il faut noter que les opérations de DESCRIPTION, MODIFICATION et SUPPRESSION commencent toujours par une opération de CITATION de l'élément à traiter.

Remarquons que les conventions précédentes indiquent qu'un élément du

système d'informations ne peut être que dans l'un des deux états distincts : DECLARE ou DECRIT (naturellement, les éléments inexistants ou supprimés n'appartiennent pas à l'application informatique à un instant donné).

Un graphe permet de préciser l'enchaînement des opérations pour changer l'état d'un élément.



Remarques :

- un nouvel élément peut être DECLARE ou CREE ;
- un élément DECLARE peut être CITE, DECRIT, MODIFIE ou SUPPRIME ;
- un élément DECRIT peut être CITE, MODIFIE ou SUPPRIME ;
- un élément MODIFIE peut être CITE, MODIFIE ou SUPPRIME ;
- un élément CREE peut être CITE, MODIFIE ou SUPPRIME ;
- un élément SUPPRIME peut être à nouveau DECLARE ou CREE.

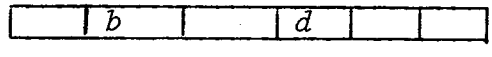
Exemple 3.6 :

Soit l'ensemble X $[A, \underline{B}, C, \underline{D}, E, F]$

On peut décrire des opérations sur des éléments x et x' ∈ X en énumérant la liste des valeurs échangées :

- déclaration de x :

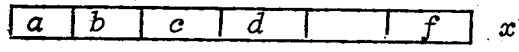
$\underline{b}$, $\underline{d}$



x

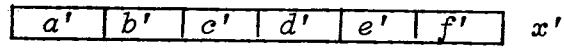
- *description de x :*

  b, a, f, c



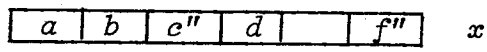
- *création de x' :*

  b', d', e', f', a, c'



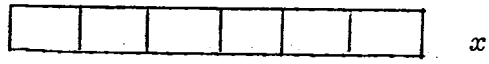
- *modification de x :*

  b, c", f"



- *suppression de x*

  b



Exemple 3.6 bis :

Soient les ensembles :

ETUDIANT [NOM, PRN, ADRESSE, NUMETUD, INSEE, ...]

BINOME (ETUDIANT, ETUDIANT) [CLEBIN, ...]

et les opérations :

(1) déclaration d'un étudiant

(dupont, jean, -, 670754, -, ...).

(2) citation et description de l'étudiant déclaré "670754"

(dupont, jean, 12 rue des lilas, 670754, 1530538130030, ...).

(3) création d'un étudiant

(dubois, marie, 1 rue des martinets, 700742, 2520669196176, ...).

(4) citation et modification d'un étudiant décrit "700742"

(dubois, marie, 3 rue des martinets, 700742, 2520669196176, ...).

(5) déclaration d'un étudiant

(durand, paul, -, 690540, -, ...).

(6) citation des étudiants "70742" et "690540" et déclaration d'un binôme

(70742 || 690540, ...).

(7) citation et suppression de l'étudiant "670754"

supprimer (dupont, jean, 12 rue des lilas, 690754, 1530538130030,...).

II.3.3 - Conséquences sur la dynamique d'une transaction d'entrée

Dans le paragraphe II.2.3, nous avons donné les principales caractéristiques des transactions (CODE, LIBELLE, DEFINITION, ENSEMBLE de REFERENCE, SENS et SCHEMA).

On peut maintenant préciser l'(les) opération(s) envisagée(s) sur des éléments dans une transaction d'entrée.

II.3.3.1 - TYPE et ELEMENT OBJET d'une transaction d'entrée

Soient : - une transaction d'entrée T,
- un ensemble E, ensemble de référence de T,
- une relation R(E,F,G).

Les actions de T peuvent être précisées :
- non seulement par des actions sur un élément de l'ensemble de référence E,
- mais aussi par des actions sur des n-uplets de la relation R,
- et/ou des actions sur des éléments des ensembles arguments de R, associés dans les n-uplets.

Ces actions peuvent être très diverses ; nous pouvons en donner une liste (non exhaustive) :

	E	R	F	G
(1)	dcl	dcl	cit	cit
(2)	dcr	dcl	cit	cit
(3)	dcr	cit	-	-
(4)	cre	cre	cit	cit
(5)	cre	dcl	cit	cit
(6)	cre	dcl	dcl	cit
(7)	cre	dcl	dcl	dcl
(8)	dcr	dcr	cit	cit
(9)	mod	cit	mod	mod
(10)	mod	sup	cit	cit
(11)	sup	-	-	-
	...	...	...	...

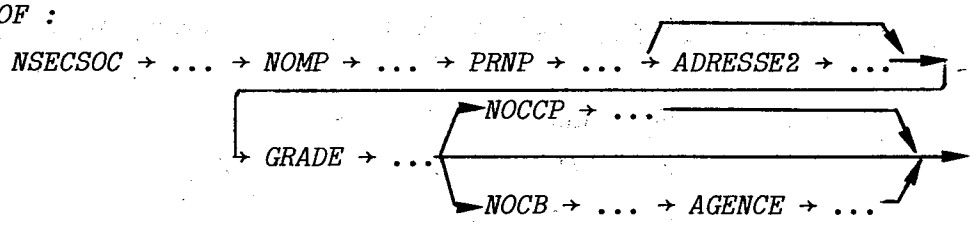
où, par exemple, l'opération (8) signifie :

- description d'un élément e de E ;
- description d'un élément r de R, r = (e,f,g) ;
- citation d'un élément f de F ;
- citation d'un élément g de G.

Exemple 3.7 :

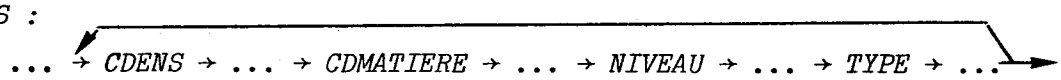
Reprenons les transactions TECREPROF et TEMODENS (exemples 2.24, 2.25).

* *TECREPROF :*



Une réalisation de cette transaction permet de CREER un élément de l'ensemble de référence ENSEIGNANT [NOMP, PRNP, NSECSOC, ADRESSE2, GRADE, ...]

* *TEMODENS :*



Une réalisation de cette transaction :

- *CITE un élément de l'ensemble de référence ENSEIGNANT*
- *CREE un élément de la relation ENSEIGNEMENT (ENSEIGNANT, MATIERE)*
- *CITE un élément de l'ensemble MATIERE.*

Convention : Soient E l'ensemble de référence d'une transaction T et une relation R(E,F,G).

Une réalisation d'une transaction T en ENTREE réalise :

- une DECLARATION,
- une DESCRIPTION,
- une CREATION,
- une CITATION,
- une MODIFICATION,
- ou une SUPPRESSION d'UN élément e de son ENSEMBLE de REFERENCE E ;

peut opérer éventuellement :

- une DECLARATION,
- une DESCRIPTION,
- une CREATION,

- une MODIFICATION,
- ou une SUPPRESSION d'UN élément r = (e,...) d'une relation R entre l'ensemble de référence et d'autres ensembles ;

ainsi que :

- les CITATIONS d'éléments d'ensembles-arguments de R, associés dans le n-uplet r = (e, ...).

"e" est appelé l'ELEMENT-OBJET de la réalisation de la transaction T et l'opération élémentaire réalisée sur l'élément objet constitue le TYPE de la transaction.

Exemple 3.8 :

La transaction TECREPROF est de TYPE "création", alors que TEMODENS est de type "citation" d'un élément de l'ensemble de référence.

r1 : NSECSOC 1490538130030 NOMP dubois PRNP jean ADRESSE2 1 rue de lilas 38000 Grenoble GRADE AS NOCCP 12125R
est une réalisation de la transaction TECREPROF, dont l'enseignant "1490538130030" est l'élément-objet qui doit être créé.

II.3.3.2 - Construction du schéma d'une transaction d'entrée

La convention adoptée précédemment quant aux actions décrites dans une transaction permet de préciser les règles de construction du schéma d'une transaction d'entrée introduites au § II.2.3.3.

Chemin simple et début d'un schéma

Nous indiquons (règle 2, § II.2.3.3) qu'un schéma de transaction commence toujours par un chemin simple. La constitution de ce chemin dépend du TYPE de la transaction d'entrée.

. Si le TYPE de la transaction d'entrée est :

- DESCRIPTION,
- CITATION,
- MODIFICATION,
- ou SUPPRESSION

ce chemin simple est uniquement composé d'un CHAMP entièrement décrit par le CODE d'une PROPRIETE-CLE de l'ENSEMBLE de REFERENCE de la transaction. C'est un *CHEMIN SIMPLE D'IDENTIFICATION.*

. Si le TYPE de la transaction est :

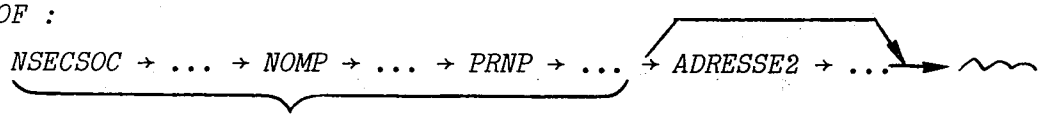
- DECLARATION,
- ou CREATION,

ce chemin simple est composé de CHAMPS qui correspondent à la liste complète des PROPRIETES de la SEMANTIQUE de DECLARATION associée à l'ENSEMBLE de REFERENCE de la transaction ; le premier champ étant décrit par une PROPRIETE-CLE de cette liste.
C'est un *CHEMIN SIMPLE de DECLARATION.*

Exemple 3.9 :

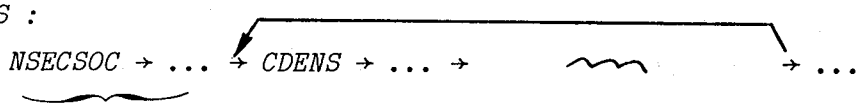
cf. les exemples 3.7 et 3.8.

- *TECREPROF :*



chemin simple de déclaration
d'un élément objet de l'ensemble de référence de TECREPROF.

- *TEMODENS :*



chemin simple d'identification
d'un élément objet de l'ensemble de référence de TEMODENS.

Chemin lié et chemin lié itératif d'un schéma

Soient une transaction T d'ensemble de référence E et une relation R(E,F,G...).
Précisons les règles 1 et 3 du § II.2.3.3.

Nous appelons chemin lié (itératif) du schéma d'une transaction T, un chemin (itératif) composé de CHAMPS décrits par des PROPRIETES AUTRES que celles de l'ENSEMBLE de REFERENCE E ; ce sont donc des propriétés d'une relation R(E,F,G,...) ou d'ensembles F,G,... arguments de R.

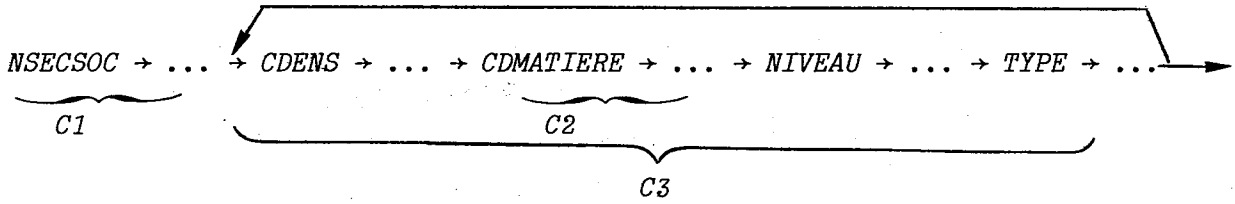
Un tel chemin est composé de champs :

- pour décrire une déclaration, description, création, modification ou suppression d'un élément r = (e,f,g,...) de R ;
- et pour citer les éléments f,g,... des ensembles arguments de R associés à l'élément objet dans le n-uplet r = (e,f,g,...).

Dans le cas d'une opération de déclaration ou de création d'un élément r(e,...) de R, ce chemin contient tous les champs permettant de décrire la sémantique de déclaration de R.

Exemple 3.10 :

Soit le schéma de la transaction TEMODENS :



C1 : chemin simple d'identification pour CITER un élément objet de l'ensemble de référence de la transaction.

C2 : chemin pour CITER un élément de l'ensemble MATIERE

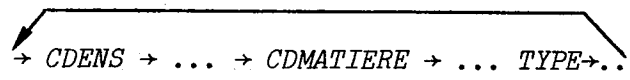
C3 : chemin lié itératif pour DECLARER des éléments de la relation ENSEIGNEMENT dont l'élément objet "cité" dans C1 et l'élément "cité" dans C2 sont les arguments.

II.3.4 - Dynamique d'une transaction de sortie

Nous avons donné (§ II.2.3) les principales caractéristiques des transactions de sortie (CODE, LIBELLE, DEFINITION, ENSEMBLE de REFERENCE et SCHEMA).

Exemple 3.11 :

Soit la transaction :
CODE : TSLISTPROF1
LIBELLE : édition des enseignements assurés par un enseignant
DEFINITION : imprimer pour enseignant : son numéro de sécurité sociale, son nom, son prénom, et les enseignements qu'il assure
SENS : sortie
ENSEMBLE de REFERENCE : ENSEIGNANT
SCHEMA :



II.3.4.1 - Type d'une transaction de sortie

On dit qu'une transaction de sortie est du TYPE :
- EDITION SIMPLE si elle réalise une EDITION relative à UN SEUL ELEMENT de l'ensemble de référence (élément-objet de la transaction) ;
- ou EDITION MULTIPLE dans le cas d'une EDITION de TOUS les ELEMENTS de l'ensemble de référence.

Exemple 3.12 :

La transaction TSLISTPROF1 (exemple 3.11) est de type "édition simple", alors que TSLISTPROF (exemple 2.25) est une "édition multiple".

II.3.4.2 - Schéma d'une transaction de sortie

La construction du schéma d'une transaction de sortie doit respecter des règles analogues à celles énoncées dans le cas d'une transaction d'entrée (cf. § II.3.3.2).

Soient : - une relation R(E, ...),
- une transaction T,
- E ensemble de référence de T.

Convention :

. Une réalisation d'une transaction de sortie T réalise :

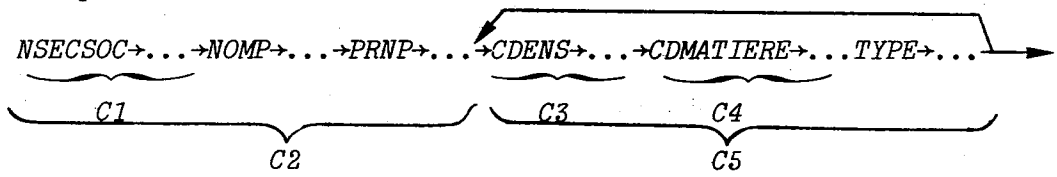
- la (les) CITATION(S) d'un (d') élément(s) e de son ensemble de référence E et des EDITIONS de valeurs de propriétés de(s) e,
- et éventuellement la (les) CITATION(S) d'un (d') élément(s) e' de la relation R ou d'ensembles-arguments de R, associé(s) à (aux) e et des EDITIONS de valeurs de(s) e'.

. Le schéma d'une transaction de sortie T commence toujours par un CHEMIN SIMPLE d'IDENTIFICATION composé du CODE d'une PROPRIETE-CLE de l'ENSEMBLE de REFERENCE pour permettre la (les) CITATION(S) de(s) e.

. Le schéma d'une transaction de sortie T peut contenir des CHEMINS LIES ou des CHEMINS LIES ITERATIFS pour CITER et EDITER le(s) élément(s) e'.

Exemple 3.13 :

- Reprenons la transaction TSLISTPROF1 :

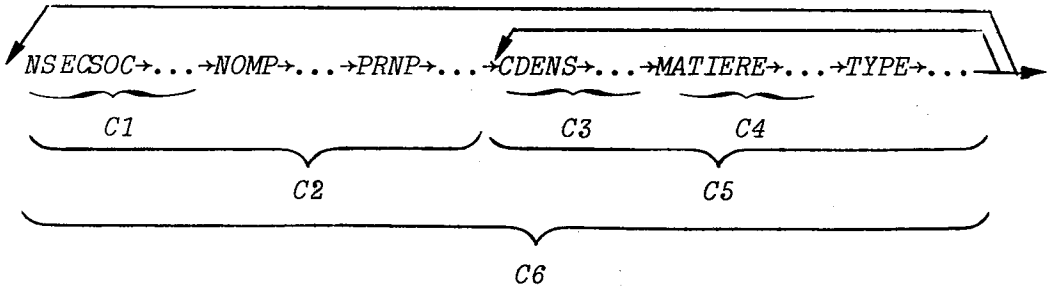


C1 = CITATION d'un enseignant (chemin simple d'identification)
C2 = chemin simple d'EDITION de valeurs de l'élément objet de la transaction
C3 = CITATION d'un enseignement
C4 = CITATION d'une matière
C5 = chemin lié itératif d'EDITION des enseignements et matières associés à un enseignant.

Une demande de réalisation de cette transaction est effectuée par la citation de l'élément objet concerné :

TSLISTPROF1 : NSECSOC 1490538130030

- Dans le cas de la transaction TSLISTPROF :



C1, C2, C3, C4 et C5 : même signification que précédemment

C6 : le schéma de la transaction TSLISTPROF est itératif, car cette transaction est du TYPE "édition multiple".

Une demande de réalisation de TSLISTPROF ne nécessite pas la citation préalable de tous les éléments objets concernés.

II.3.5 - Gestion de la valeur d'une propriété d'un élément

Nous avons indiqué (§ II.2.1.2) que la valeur d'une propriété d'un élément d'un ensemble est unique et appartient à l'espace des valeurs de cette propriété ou est une valeur particulière (u ou RAS).

Les opérations autorisées (déclaration, description, ...) sur les éléments d'un système d'informations concernent aussi la GESTION des VALEURS de leurs propriétés ; c'est-à-dire les modalités d'ELABORATION, de STOCKAGE et/ou d'UTILISATION de ces valeurs;

Convention : Nous appelons MODE d'une propriété les diverses formes de gestion des valeurs pour les éléments.

Le MODE d'une propriété est décrit par un texte explicatif précisant si :

- l'ELABORATION des valeurs est réalisée par une SAISIE d'informations ou par un CALCUL interne ;
- le STOCKAGE des valeurs est DEFINITIF ou TEMPORAIRE ;
- l'UTILISATION est EXTERNE (EDITION) ou INTERNE (utilisée pour des calculs et contrôles internes).

Exemple 3.14 :

LIBELLE DE PROPRIETES	*ELABORATION*		*STOCKAGE*		*UTILISATION*	
	SAISIE	*CALCULEE*	*DEFINITIF*	*TEMPORAIRE*	*EXTERNE*	*INTERNE*
pour un ETUDIANT :						
- date de naissance	*		*		*	*
- âge		*		*	*	
pour une SALLE :						
- code de la salle	*		*		*	
- code bâtiment (compris dans le code salle)	*			*	*	*
pour un ENSEIGNEMENT (ENSEIGNANT,MATIERE)						
- durée annuelle	*		*		*	*
pour un ENSEIGNANT						
- nombre total annuel d'heures d'enseign.		*	*		*	

II.3.6 - Bilan sur les descriptions statiques et dynamiques du modèle

Dans les § II.2 et II.3, nous avons présenté les concepts essentiels d'un modèle fonctionnel des données et des traitements d'une application informatique ; avant de proposer des extensions de ce modèle, nous le résumons dans le tableau suivant :

MODELE FONCTIONNEL d'APPLICATION	ENSEMBLES de BASE	RELATIONS	PROPRIETES	TRANSACTIONS
DEFINITION	ensemble d'éléments concrets	ensemble de liaisons entre éléments d'ensembles	propriété attachée aux éléments d'un ensemble	opération automatisée d'échange d'informations
Spécifications fondamentales et structurelles	- CODE - 1 ou plusieurs CLES - 1 liste de PROPRIETES - 1 SEMANTIQUE de DECLARATION des éléments	- CODE - DEGRE : 2 à n - ARGUMENTS et FCT-DEPENDANCE, MIN, ~~MAX, pour chaque~~ argument - 1 ou plusieurs CLES - 1 liste de PROPRIETES - 1 SEMANTIQUE de DECLARATION des éléments	- CODE - ESPACE des VALEURS - COMPOSITION éventuelle - MODE (saisie, calculée, ...)	- CODE - SENS (E, S, E/S) - ENSEMBLE de REFERENCE - TYPE de l'opération (decl, decr, cre, cit, mod, sup, edt simple, edt multiple) - SCHEMA de transaction
Spécifications qualitatives et documentaires	- LIBELLE - DEFINITION - TAILLE	- LIBELLE - DEFINITION - TAILLE	- LIBELLE - DEFINITION	- LIBELLE - DEFINITION

Modèle fonctionnel des données et des transactions d'une application informatique

II-4 - EXTENSIONS DIVERSES DU MODELE

II.4.1 - Extensions du modèle des données

II.4.1.1 - TYPE d'une propriété

Définition :

Le TYPE d'une propriété précise la nature de son ESPACE de VALEURS.

Dans tous les langages de programmation, cette notion existe.

Exemple 4.1 :

En COBOL, on écrit : *NOM PICTURE A(20)*
ADRESSE PICTURE X(60)
AGE PICTURE 9(2)

En SOCRATE : *NOM mot 15*
ADRESSE texte 1
AGE de 10 à 99

Convention :

Pour chaque propriété, nous indiquerons son TYPE qui peut être l'un des suivants :

- CODE : Les valeurs de ce type sont des chaînes de 15 caractères au maximum, alphabétiques, numériques ou alphanumériques, sans espaces.
- MOT : Ce sont des chaînes, d'un maximum de 30 caractères quelconques alphabétiques, numériques ou spéciaux (tiret, apostrophe, ...).
- NUMERIQUE ENTIER BORNE : Valeurs numériques qui représentent des entiers positifs, nuls ou négatifs appartenant à un ou plusieurs intervalles fermés définis par leurs bornes.
- NUMERIQUE REEL BORNE : Nombre réel dont la valeur peut appartenir à un ou plusieurs intervalles fermés définis par leurs bornes.
- LISTE DE CODES : Liste de valeurs de type CODE.
- LISTE DE MOTS : Liste de valeurs de type MOT.
- LISTE D'ENTIERS : Liste de valeurs numériques entières.
- LISTE DE REELS : Liste de valeurs numériques réelles.

. LIGNE : Chaîne de caractères alphanumériques comprenant au maximum 60 caractères.

. TEXTE : Suites de lignes.

Une liste correspond à un espace de valeurs qui ne change pas au cours du temps.

Le type CODE (ou liste de CODES) est le type obligatoire de toute PROPRIETE-CLE des ensembles. Nous autorisons un maximum de 15 caractères, sachant que les "codes" sont en général courts (6 à 8 caractères), sauf certains qui ont plus de 10 caractères (numéro INSEE sur 13 caractères).

Exemple 4.2 :

Soient les ensembles :

ETUDIANT [NOM, INSEE, AGE, ADRESSE, ...]
SALLE [CDS, ETAGE, CAPACITE, ...]
MATIERE [CDMATIERE, TITRE, ...]

et les types des propriétés avec des exemples de valeurs :

- *NOM : "mot"*
  Dupont-d'aubigny, Duroc
- *INSEE : "code"*
  1490538130030
- *AGE : "numérique entier borné" (de 15 à 99)*
  18, 21
- *ADRESSE : "texte"*
  Belvédère 2
  10, rue des fleurs
  38170 - SEYSSINS-LA-PLAINE
- *CDS : "code"*
  C108, B309
- *ETAGE : "liste d'entiers" = {1,2,3}*
  1, 3
- *CAPACITE : "numérique entier borné" (de 1 à 500)*
  45, 18

- *TITRE : "ligne"*

  les systèmes de bases de données : concepts et utilisations

- *CDMATIERE : "liste de codes" {maths, gestion, anglais, informatique, économie}*

  anglais, économie.

Remarques méthodologiques :

a) Nous pouvons justifier le choix de ces divers TYPES, en faisant remarquer que ces distinctions sont nécessaires pour des raisons algorithmiques ou pratiques :

- certaines propriétés ont des valeurs qui ne peuvent être que lues ou écrites sans possibilité d'effectuer aucune opération de calcul ni de comparaison (types "ligne" et "texte") ;
- d'autres propriétés ont des valeurs utilisables dans des opérations de calcul arithmétique (types "numérique entier borné", "numérique réel borné", "liste d'entiers" et "liste de réels") ;
- des valeurs de propriétés permettent d'effectuer des comparaisons (tous les types sauf "ligne" et "texte") ;
- il est pratique de distinguer un type "ligne", pour décrire des libellés ou de brèves explications (titre d'une matière, ...), d'un type "texte" pour rassembler des suites de mots très importantes (adresse, résumé du contenu d'un cours, ...).

b) Nous aurions pu distinguer d'autres types de valeurs, par exemple :

- en séparant le type "CODE" en trois autres types qui sont les codes alphabétiques, les codes numériques et les codes alphanumériques. Cette distinction était inutile, étant donné que les valeurs de type CODE sont essentiellement manipulées par des algorithmes de comparaison, de recherche ou de tri souvent indépendants de la nature (numérique ou littérale) des caractères.

  On peut remarquer que la nature alphanumérique des codes apparaît généralement dans le cas des propriétés-clés composées à aspect mnémonique (code d'une salle : C 1 08)

  C : code bâtiment ; 1 : étage ; 08 : numéro

- en distinguant un type ADRESSE des types LIGNE ou TEXTE, ou un type DATE (JOUR, MOIS, ANNEE) du type NUMERIQUE ENTIER BORNE. ADRESSE et DATE étant des notions qui apparaissent fréquemment dans les applications en informatique de gestion.

Des distinctions de types plus ou moins fines n'ont d'intérêt que pour autant qu'elles expriment des contrôles-standards intéressants. Au niveau fonctionnel où nous nous sommes placés, ces descriptions de données ne sont pas proposées pour réserver des zones sur une mémoire magnétique.

c) Le type d'une propriété composée est facultatif, sauf si cette propriété est une CLE, car il doit être alors "code" ou "liste de codes".

II.4.1.2 - Les sous-ensembles

Considérons un ensemble (ensemble de base ou relation) E

$$E\ [P_1:V_1,\ P_2:V_2,\ \ldots,\ P_i:V_i,\ \ldots\ P_n:V_n]$$

Définition :

Nous définissons un SOUS-ENSEMBLE SE de l'ensemble E de la manière suivante :

SE est un ENSEMBLE : $SE \in \mathcal{P}(E)$,

$$SE\ [P'_1:V'_1,\ \ldots\ P'_i:V'_i,\ \ldots\ P'_n:V'_n,\ P_{n+1}:V_{n+1},\ \ldots\ P_{n+k}:V_{n+k}]$$

et SE vérifie les trois conditions :

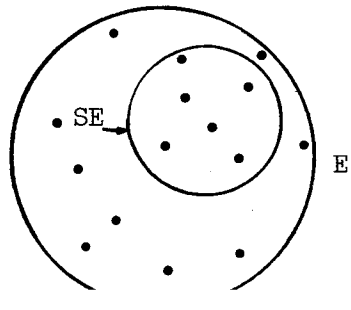
a) $\forall i = 1,2,\ldots n$

et $\forall e \in SE$, alors $P_i(e) = P'_i(e)$ et $P'_i(e) \in V'_i \subseteq V_i$

b) $\exists j = 1,2,\ldots$ ou $n : V'_j \subsetneq V_j$

P'_j est une restriction de l'application P_j

c) $k \geq 0$, il peut exister des propriétés nouvelles définies pour SE (notées $P_{n+1}, \ldots P_{n+k}$)



Nous pouvons dire qu'un sous-ensemble SE de E est défini

- par une ou plusieurs PROPRIETES-SUPPORTS de E
- et (ou) par une ou plusieurs RELATIONS-SUPPORTS dont E est argument.

* Une PROPRIETE-SUPPORT est une propriété P de E (P : E→V) telle que
  ∀e ∈ SE
  P' propriété de SE, P(e) = P'(e) et P'(e) ∈ V' ⊆ V
  P, propriété de E, admet une restriction P' (P' : SE→V'), propriété de SE.

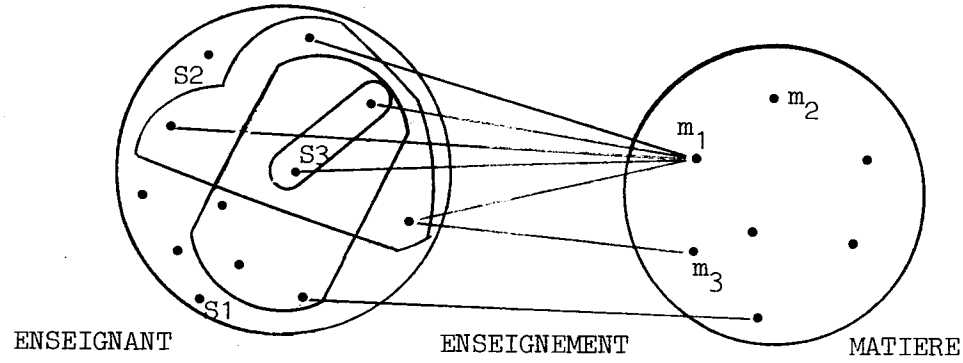
* Une RELATION-SUPPORT est une relation R(E,F,G...) dont E est argument telle que e ∈ SE <==> (e,f,g,...) ∈ R
  avec e ∈ E, f ∈ F, g ∈ G, ...

Exemple 4.3 :

Soient les ensembles ENSEIGNANT [NSECSOC, NOMP, PRNP, GRADE : {P, MC, MA, A, V}, ...]

MATIERE [CDMATIERE, ...]

et la relation ENSEIGNEMENT (ENSEIGNANT, MATIERE) [CDENS, ...]



On peut définir les sous-ensembles :

S1 : . ensemble des enseignants qui ont le grade d'assistant (A)
. GRADE est une PROPRIETE-SUPPORT

S2 : . ensemble des enseignants qui enseignent la matière m1
. ENSEIGNEMENT est une RELATION-SUPPORT

S3 : . ensemble des "assistants" qui enseignent la matière m1
. GRADE est une PROPRIETE-SUPPORT
. ENSEIGNEMENT est une RELATION-SUPPORT

Caractéristiques d'un sous-ensemble

Les sous-ensembles sont des ensembles que nous caractérisons comme les ensembles de base (cf. II.2.1.1, II.2.1.4, II.3.2) par :

- un CODE d'identification discriminant,
- un LIBELLE explicatif,
- une DEFINITION,
- une TAILLE,
- une ou plusieurs PROPRIETES-CLES,
- une liste de PROPRIETES,
- une SEMANTIQUE de DECLARATION.

Nous pouvons aussi décrire :

- des opérations élémentaires de déclaration, description, création, citation, modification et suppression de leurs éléments ;
- des transactions dont ils sont les ensembles de référence.

Ces descriptions doivent être complétées par la définition des PROPRIETES-SUPPORTS de la PARTITION associées au sous-ensemble afin de préciser la (les) propriété(s)-support(s) et (ou) la (les) relation(s)-support(s).

Notation d'un sous-ensemble

Les propriétés-supports sont marquées d'une étoile et l'ensemble englobant est cité entre parenthèses ainsi qu'éventuellement les relations supports.

Exemple 4.4 :

- *S1 sous-ensemble des assistants*

  S1(ENSEIGNANT) [*NSECSOC, NOMP, PRNP, GRADE* : {A}, ...*]

- *S2 sous-ensemble des enseignants de la matière m1*

  S2(ENSEIGNANT/CDMATIERE : {m1} dans ENSEIGNEMENT)* [*NSECSOC,NOMP,...*]

- *S3 sous-ensemble des assistants qui enseignent la matière m1*

  S3(ENSEIGNANT/CDMATIERE : {m1} dans ENSEIGNEMENT)* [*NSECSOC, ...*
  GRADE : {A}, ...*]

- *S4 sous-ensemble des enseignements en première année en TD (travaux dirigés) et TP (travaux pratiques)*

  S4(ENSEIGNEMENT) [*CDENS, NIVEAU* : {1}, TYPE* : {TD,TP}, ...*]

Remarques méthodologiques :

a) Il est évident que la TAILLE d'un sous-ensemble SE de E doit être inférieure à la taille de E.

b) Les PROPRIETES de E sont aussi des propriétés de tout sous-ensemble SE de E ; mais il est possible de décrire éventuellement des PROPRIETES spécifiques d'un sous-ensemble qui ne sont pas propriétés de l'ensemble englobant.

c) Une PROPRIETE-CLE d'un ensemble E est toujours propriété-clé de tout sous-ensemble de E, mais un sous-ensemble SE de E peut avoir une propriété-clé qui lui est propre et qui n'est pas propriété-clé de E.

d) On ne peut "DECLARER" ou "CREER" un élément d'un sous-ensemble SE de E que si l'on assure aussi son existence dans E ; aussi, la SEMANTIQUE de DECLARATION d'un SOUS-ENSEMBLE SE de E comporte-t-elle :
   - la sémantique de déclaraton de E,
   - les propriétés supports de la partition,
   - plus éventuellement des propriétés spécifiques du sous-ensemble SE.

e) La définition d'un sous-ensemble avec plusieurs propriétés-supports et/ou plusieurs relations-supports constitue une INTERSECTION de sous-ensembles décrits chacun par une seule propriété-support ou relation-support alors que la définition d'un sous-ensemble avec une seule propriété-support dont l'espace de valeurs réduit contient plus d'une valeur est une UNION de sous-ensemble.

f) Un sous-ensemble peut être lui-même considéré comme un ensemble : et ainsi, il est possible de définir :
   - des sous-ensembles d'un sous-ensemble
   - des relations dont certains arguments sont des sous-ensembles.

g) L'introduction du concept de sous-ensemble dans notre modèle des données suppose une analyse descendante des données : il faut d'abord décrire les ensembles les plus généraux, puis en caractériser des parties appelées sous-ensembles.

h) Un sous-ensemble peut être l'ensemble de référence d'une transaction (cf. § II.2.3.3).

Exemple 4.5 :

Soit un ensemble de base PERSONNE regroupant toutes les personnes en activité à l'IUT :

PERSONNE [INSEE, NOM, ADRESSE, ACTIVITE : {PROF, ELEV, ADM},...]

où l'ACTIVITE d'une PERSONNE est d'enseigner (PROF), d'étudier (ELEV) ou d'administrer (ADM).

On peut définir les sous-ensembles :

ETUDIANT (PERSONNE) [INSEE, NOM, ADRESSE, ACTIVITE :{ELEV},ANNEE:{1,2,AS}, NUMETUD, ...]*

activité : propriété-support de la partition
année : propriété propre au sous-ensemble ETUDIANT
numetud : propriété-clé propre à ETUDIANT
(remarques b et c)

ENSEIGNANT (PERSONNE) [INSEE, NOM, ADRESSE, ACTIVITE :{PROF},STATUT:{P,MC,MA, A,V}, MATIERE : {M,I,A,G,E,T}, ...]*

Des sous-ensembles de sous-ensembles (remarque f) :

. *les étudiants de première année*
ETUD1 (ETUDIANT) [INSEE, NOM, ADRESSE, ACTIVITE : {ELEV}, ANNEE* : {1}, NUMETUD, GROUPE, ...]*

. *les assistants*
ASST (ENSEIGNANT) [INSEE, ... STATUT : {A}, MATIERE : {M,I,A,G,E,T}, ...]*

. *les enseignants de maths*
PMAT (ENSEIGNANT) [INSEE, ... STATUT : {P,MC,MA,A,V}, MATIERE : {M}, ...]*

. *les assistants de maths*
ASSMAT (ENSEIGNANT) [INSEE, ... STATUT : {A}, MATIERE* : {M}, ...]*

$$ASSMAT = ASST \cap PMAT$$
$$= \{e \in ENSEIGNANT : STATUT(e) = "A" \text{ et } MATIERE(e) = "M"$$

(remarque e)

. *les vacataires*
VACT (ENSEIGNANT) [INSEE, ... STATUT : {V}, ...]*

. *les vacataires ou assistants*

ASSTVACT (ENSEIGNANT) [*INSEE*, ... *STATUT** : {*A,V*}, ...]

ASSVACT = ASST ∪ VACT

= {e ∈ ENSEIGNANT : STATUT(e) = "A" ou STATUT(e) = "V"}

(remarque e)

Des sous-ensembles arguments de relation (remarque f)

COURS (ETUDIANT, ENSEIGNANT, SALLE) [*CDCOURS*, *FORME* : {*TP,TD,C*}, ...]

Des sous-ensembles de relations :

ENSTP (COURS) [*CDCOURS*, *FORME** : {*TP*}, ...]

Remarques méthodologiques

i) Un élément peut-il appartenir à plusieurs sous-ensembles d'un même ensemble ?

Dans le formalisme que nous proposons, il est possible de prévoir l'appartenance d'un élément à un ou plusieurs sous-ensembles d'un même ensemble. Mais ceci doit être pris en compte dès la description du modèle des données de l'application concernée, ce qui n'est pas étonnant !

Exemple 4.6 :

Structure 1 :

Soient l'ensemble : PERSONNE [*INSEE*,... *ACTIVITE* : {*prof, elev, adm*}, ...]
et les sous-ensembles : ENSEIGNANT (PERSONNE) [*INSEE*, ... *ACTIVITE** :
{*prof*}, *statut*, ...]
ETUDIANT (PERSONNE) [*INSEE*, ... *ACTIVITE** :
{*elev*}, *NUMETUD*, ...]]

Avec la règle d'unicité de valeur d'une propriété pour un élément donné, la structure 1 interdit le fait qu'une personne puisse être ETUDIANTE et ENSEIGNANTE, et ainsi avoir des propriétés particulières en tant qu'étudiante et en tant qu'enseignante.

$$ENSEIGNANT \cap ETUDIANT = \emptyset$$

Ce problème serait résolu avec la structure 2 :

Structure 2 :

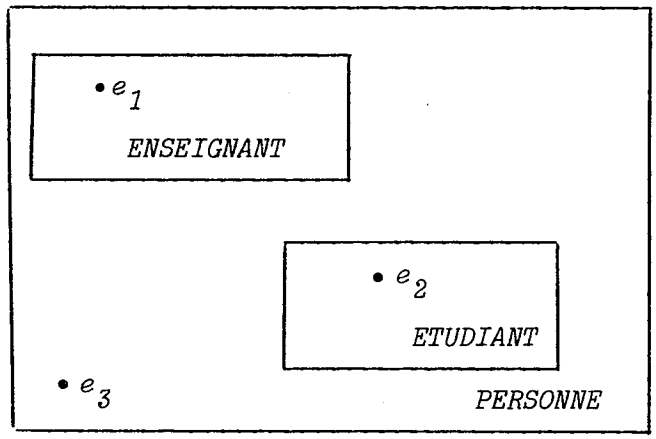
PERSONNE2 [*INSEE*,... *ACTIVITE* : {*prof,elev,prof-elev,adm*}, ...]
ENSEIGNANT2 (PERSONNE2) [*INSEE*,... *ACTIVITE** : {*prof,prof-elev*}, *statut*,...]

ETUDIANT2 (PERSONNE2) [INSEE,... ACTIVITE : {elev,prof-elev}, NUMETUD,...]*

Ainsi, avec cette deuxième structure
ENSEIGNANT2 ∩ ETUDIANT2 ≠ ∅ "si des personnes ont l'activité = "prof-elev"

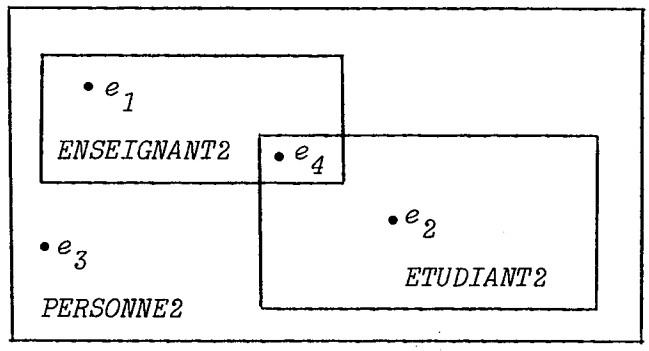
Une même personne peut alors être enseignante et étudiante et donc avoir "un statut", un "numetud", ...

Modèle 1



ACTIVITE (e_1) = 'prof'
ACTIVITE (e_2) = 'elev'
ACTIVITE (e_3) = 'adm'

Modèle 2



ACTIVITE (e_1) = 'prof'
ACTIVITE (e_2) = 'elev'
ACTIVITE (e_3) = 'adm'
ACTIVITE (e_4) = 'prof-elev'

j) Différences de nature entre les ensembles de base, les relations et les sous-ensembles

Les distinctions possibles entre ces divers types d'ensembles sont essentiellement introduites par des contraintes sémantiques relatives à la dynamique des opérations définies sur les éléments de ces ensembles.

Exemple 4.7 :

Soient les deux problèmes :

- *affectation d'étudiants dans des groupes de niveaux pour des cours de rattrapage*

- *répartition des étudiants dans des locaux.*

On pourrait définir deux structures logiques différentes pour représenter les données relatives aux deux problèmes :

Structure 1 : - *ensembles de base : ETUDIANT [..., GNIV : {1,2,...n}, ...]*
SALLE [...]
- *une relation REPART (ETUDIANT, SALLE) [...]*
- *sous-ensembles G1 (ETUDIANT) [... GNIV* : {1}, ...]*
G2 (ETUDIANT) [... GNIV : {2}, ...]*
...

Structure 2 : - *ensembles de base : ETUDIANT [...]*
SALLE [...]
GNIV [...]
- *relations : REPART (ETUDIANT, SALLE) [...]*
AFFECT (ETUDIANT, GNIV) [...]

Nous pouvons remarquer les différences sémantiques entre ces deux structures proposées :

- *dans la structure 1, la suppression de l'ensemble "ETUDIANT" implique des suppressions de la relation "REPART" et des sous-ensembles G1, G2, ..., mais il existe encore des salles dans l'établissement*
- *alors que dans la structure 2, cette suppression de l'ensemble "ETUDIANT" implique la suppression des deux relations "REPART" et "AFFECT", mais les salles et, ce qui paraît illogique, les groupes de niveau existent toujours, alors qu'il n'y a plus d'étudiants.*

Aussi, est-il conseillé d'adopter, pour un tel problème, la structure 1.

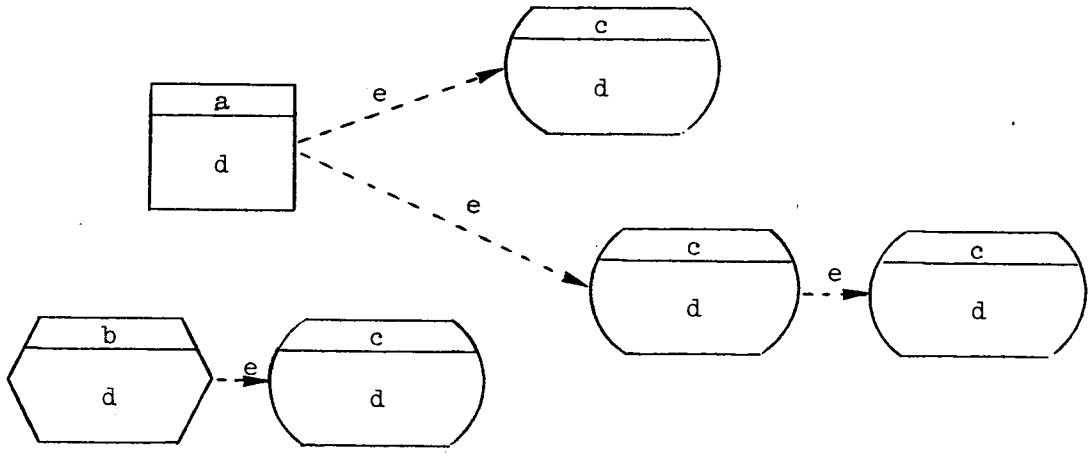
Les choix entre des définitions d'ensembles de base, de relations et/ou de sous-ensembles ne sont possibles que si l'on connaît avec assez de précision les règles logiques à respecter lors d'opérations de création ou de suppression d'éléments de ces ensembles.

Mais nous ne conseillons la définition d'un sous-ensemble S d'un ensemble E que si au moins l'une des questions suivantes admet une réponse positive :

- S a-t-il des propriétés nouvelles par rapport à E ?
- S peut-il être ensemble de référence d'au moins une transaction ?

Représentation graphique d'une structure de données contenant des sous-ensembles

Complétons le symbolisme introduit § II.2.2.5.



a : CODE d'un ensemble de base

b : CODE d'une relation

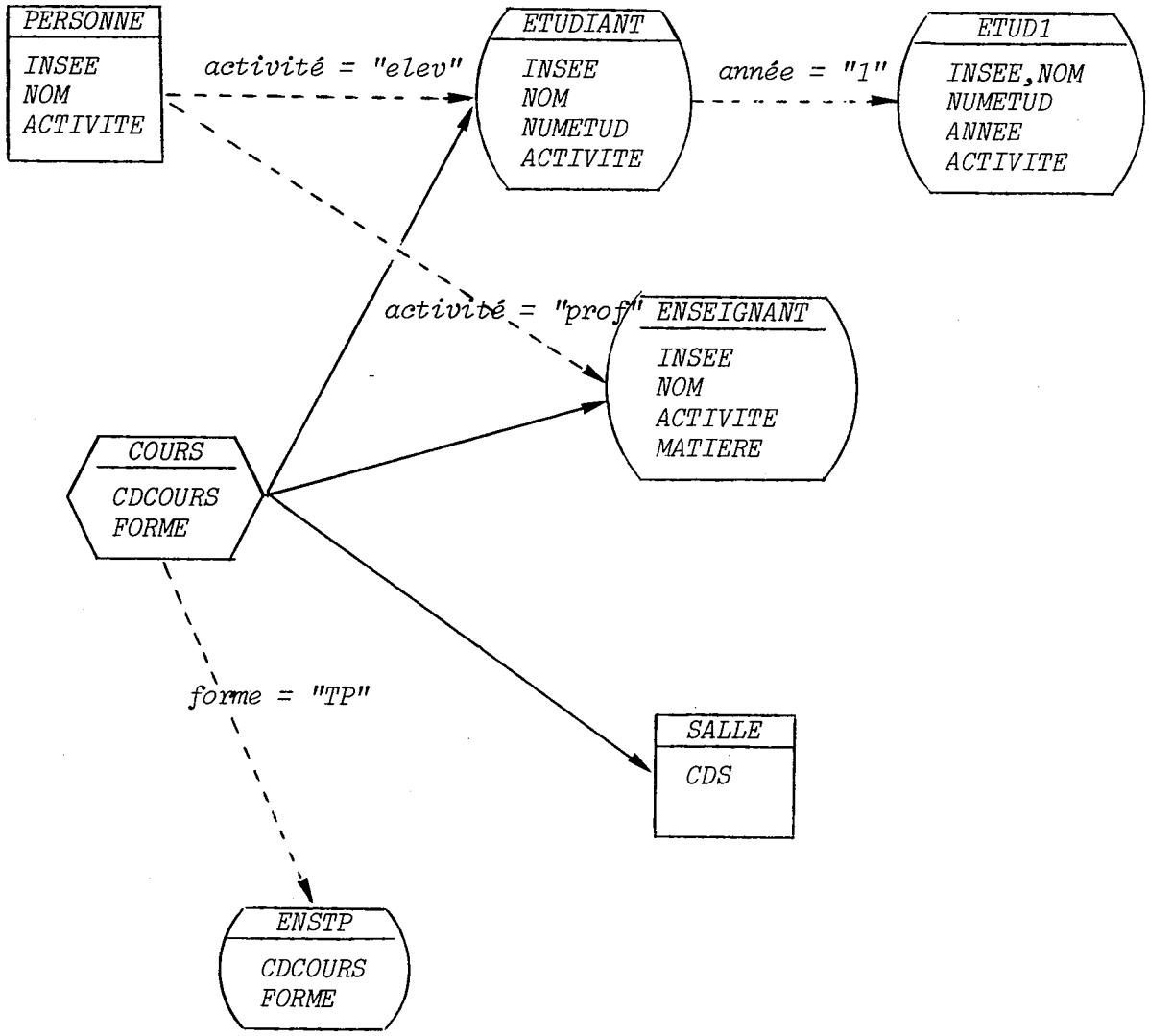
c : CODE d'un sous-ensemble d'ensemble de base, de relation ou de sous-ensemble

d : CODES des propriétés de la sémantique de déclaration de l'ensemble, plus éventuellement d'autres CODES de propriétés de l'ensemble

e : CODES des propriétés supports de la partition

Exemple 4.8 :

Décrivons graphiquement les ensembles définis dans l'exemple 4.5.



II.4.1.3 - Les agrégats

Les ensembles (ensembles de base, sous-ensembles ou relations) définis dans les paragraphes précédents regroupent des éléments caractérisés par des valeurs de propriétés.

Soient un ensemble E

et une propriété P de E ; P : E → V

e ∈ E → P(e) ∈ V

e est un élément

Mais ces notions deviennent insuffisantes si l'on désire maintenant s'intéresser aux applications produisant en sortie des informations de contrôle de gestion ou de statistiques.

Dans de telles applications, les transactions d'entrée gèrent les élément les plus atomiques par création, modification, ... ou suppression (exemples inscription d'un élève, mise à jour de la situation administrative d'un enseignant), alors que les transactions de sortie caractérisent des agrégats de ces éléments atomiques (exemples : éditer la moyenne d'âge d'un groupe d'élèves, éditer le nombre moyen annuel d'heures d'enseignement d'un professeur dans une matière).

Aussi est-il nécessaire dans de nombreux cas de permettre un découpage d'un ensemble E en classes d'équivalence S (sous-ensembles appartenant à $\mathcal{P}(E)$ pour lesquelles on peut définir de nouvelles propriétés $\mathbb{P}$.

C'est-à-dire décrire des propriétés $\mathbb{P}$ telles que :

$\mathbb{P}(S) \in \mathbb{V}$ où S est un ensemble

Nous dirons que : . l'ENSEMBLE S est un ELEMENT d'un AGREGAT $\mathcal{A}$

. $\mathbb{P}$ est une PROPRIETE de l'AGREGAT $\mathcal{A}$:

$$\mathbb{P} : \mathcal{A} \rightarrow \mathbb{V}$$
$$S \in \mathcal{A} \rightarrow \mathbb{P}(S) \in \mathbb{V}$$

Définition :

Un agrégat est un ENSEMBLE d'ENSEMBLES.

Exemple 4.9 :

Soient les ensembles :

ENSEIGNANT [NOMP, MATIERE : {M1,M2,M3}, STATUT : {P,MC,MA,A,V},...

ETUDIANT [NOM, GROUPE : {1,2,3,4}, ...]

et les sous-ensembles :

EQUIP1 (ENSEIGNANT) [NOMP, MATIERE : {M1}, ...]*

EQUIP2 (ENSEIGNANT) [NOMP, MATIERE : {M2}, ...]*

EQUIP3 (ENSEIGNANT) [NOMP, MATIERE : {M3}, ...]*

GROUP1 (ETUDIANT) [NOM, GROUPE : {1}, ...]*

GROUP2 (ETUDIANT) [NOM, GROUPE : {2}, ...]*

GROUP3 (ETUDIANT) [NOM, GROUPE : {3}, ...]*

GROUP4 (ETUDIANT) [NOM, GROUPE : {4}, ...]*

* *On peut considérer l'ensemble ENSEIGNANT comme l'élément unique d'un agrégat PROF.*

Ce nouvel ensemble (agrégat) PROF a les propriétés :

NB = nombre effectif d'enseignants de l'année en cours,
(≠ TAILLE d'ENSEIGNANT qui est le nombre maximum)
NBM1 = nombre d'enseignants de la matière M1,
NBVCT = nombre d'enseignants vacataires,
NBAM1 = nombre d'assistants qui enseignent la matière M1,
...

* *On peut construire l'agrégat EQUIPE regroupant les 3 éléments (3 sous-ensembles) EQUIP1, EQUIP2 et EQUIP3.*

A l'agrégat EQUIPE, on peut associer les propriétés :

CDEQP : code d'une équipe,
NBPRF : nombre d'enseignants dans une équipe,
NRESP : nom du responsable de l'équipe,
...

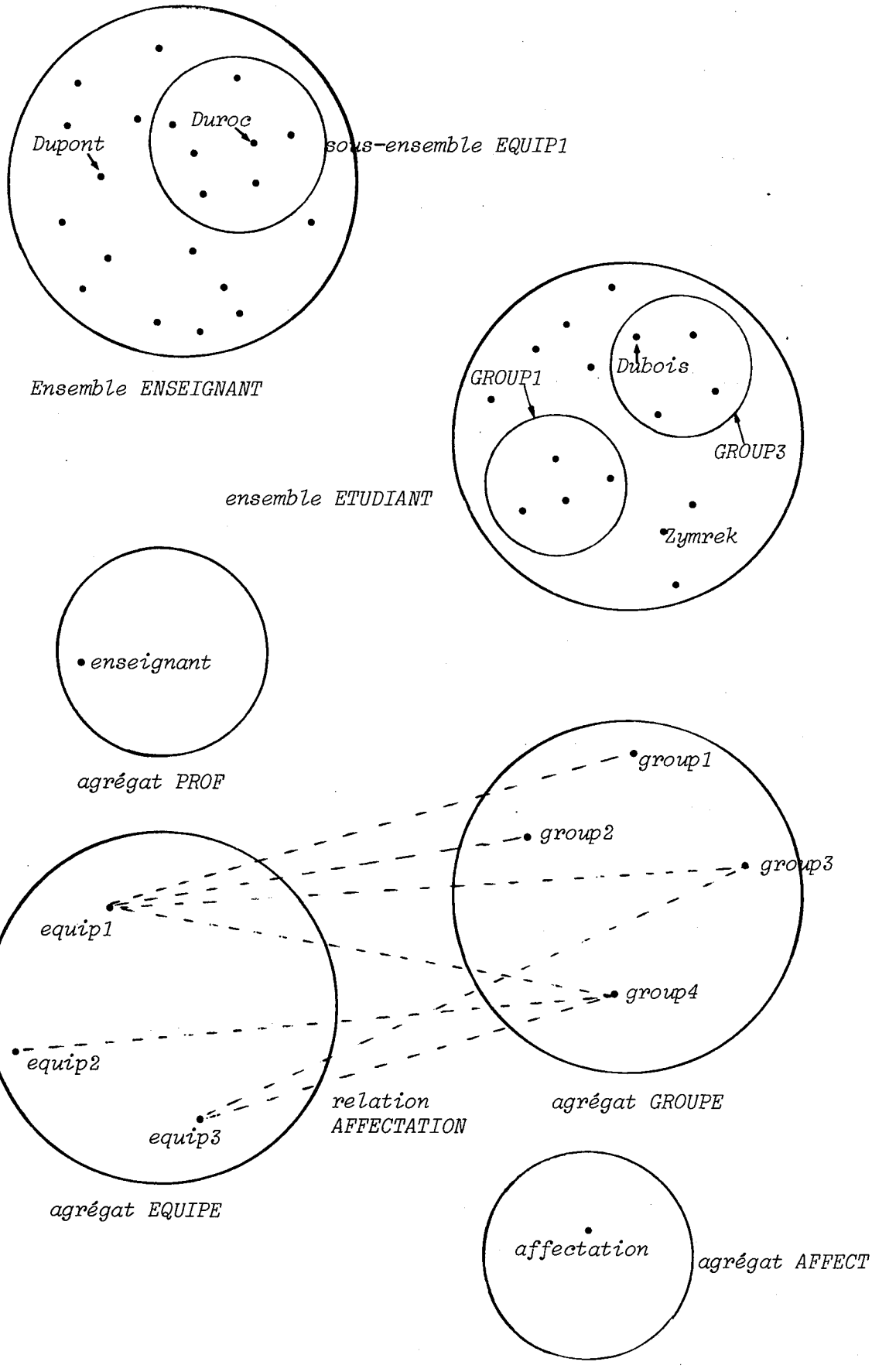
* *Soit l'agrégat GROUPE composé des 4 éléments GROUP1, GROUP2, GROUP3 et GROUP4 et les propriétés :*

NBETG : nombre d'étudiants d'un groupe,
DELG : nom du délégué du groupe,
...

* *Il est possible de décrire une relation AFFECTATION avec les arguments agrégats EQUIPE et GROUPE:*

AFFECTATION (GROUPE, EQUIPE) [DURTOT, DURJR, ...]
DURTOT : durée totale d'enseignement par une équipe à un groupe
DURJR : durée journalière maximale supportée par un groupe avec une équipe
...

ainsi qu'un agrégat (AFFECT) de la relation AFFECTATION et les propriétés :
DURM : durée moyenne annuelle d'enseignement d'une équipe à un groupe
DURJM : durée journalière moyenne d'enseignement supportée par un groupe avec une équipe.



Exemples d'agrégats

Remarques méthodologiques

- Généralement on utilise une DEFINITION en EXTENSION (cf. II.2.1.4) pour décrire un agrégat :

  PROF = {enseignant}
  EQUIPE = {equip1, equip2, equip3}
  GROUPE = {group1, group2, group3, group4}.

- On peut construire l'agrégat
  . d'un ensemble (cf. PROF)
  . d'une relation (cf. AFFECT)
  . de plusieurs sous-ensembles (cf. EQUIPE, GROUPE).

- Il est possible d'établir des relations entre des agrégats (cf. AFFECTATION).

- On construit généralement un agrégat pour décrire les données (plus synthétiques) relatives à une transaction de sortie ou certaines fois pour une transaction d'entrée.

- Un agrégat A peut être l'ensemble de référence d'une transaction d'entrée ou de sortie (cf. § II.2.3.3).

Conventions

- Un AGREGAT est un ENSEMBLE A qui peut être caractérisé par un CODE, un LIBELLE, une DEFINITION, une TAILLE, des PROPRIETES P_i, et il peut être désigné comme ensemble de référence d'une transaction.

- Un AGREGAT A peut être composé :
  . d'un élément unique qui est
    - un ensemble de base E
    - une relation R
    - un sous-ensemble S d'un ensemble de base E ou d'une relation R
  . ou de plusieurs éléments qui sont des sous-ensembles d'un même ensemble de base E ou d'une même relation R.

L'ensemble $\mathcal{E}$ (ensemble de base E ou relation R) est appelé l'ENSEMBLE de REFERENCE de l'AGREGAT A que nous notons :

$$A/\mathcal{E}[P_1, P_2, \ldots P_n] = \{S_1, S_2, \ldots S_k\}$$
$$\forall S_i \in A \qquad S_i \in \mathcal{P}(\mathcal{E})$$

Les éléments d'un agrégat sont des éléments de l'ensemble des parties de son ensemble de référence.

Exemple 4.10 :

Donnons les notations des agrégats décrits dans l'exemple 4.9.

PROF/ENSEIGNANT [NB, NBM1, NBVCT, NBAM1, ...] = {enseignant}
EQUIPE/ENSEIGNANT [CDEQP, NBPRF, NRESP, ...] = {equip1,equip2, équip3}
GROUPE/ETUDIANT [NBETG, DELG, ...] = {group1, group2, group3, group4}
AFFECT/AFFECTATION [DURM, DURJM, ...] = {affectation}

Les ensembles ENSEIGNANT, ETUDIANT et AFFECTATION sont des ensembles de référence des divers agrégats décrits.

Valeurs particulières de la clé d'identification d'un élément d'un agrégat

Pour tout agrégat composé de plus d'un élément, il est obligatoire de définir une propriété-clé pour identifier ses éléments.

Soient un agrégat A sur un ensemble de référence $\mathcal{E}$ et $E_1, E_2, \ldots E_n$ les éléments de A ($E_i \in \mathcal{P}(\mathcal{E})$).

En général, pour chacun des éléments E_i, la clé d'identification a une valeur v_i prédéfinie qui est le code de l'ensemble E_i devenu élément de l'agrégat.

De plus, si les ensembles E_i ont été définis comme sous-ensembles de $\mathcal{E}$ sur une "propriété support" P_x de type liste de valeurs, alors la valeur v_i ci-dessus est une valeur de cette liste définissant l'espace des données de P_x.

Exemple 4.11 :

Soient l'ensemble de base
ENSEIGNANT [..., SPECIALITE : {mathématicien, informaticien, angliciste,...

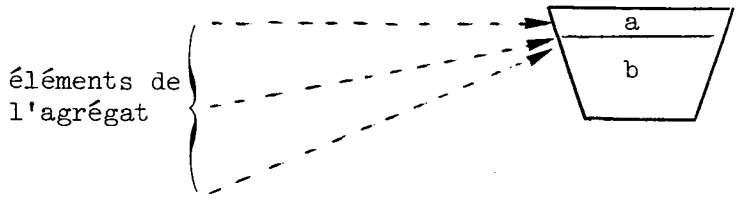
les sous-ensembles :
MATHEMATICIEN (ENSEIGNANT) [..., SPECIALITE : {mathématicien}, ...]*
INFORMATICIEN (ENSEIGNANT) [..., SPECIALITE : {informaticien}, ...]*
ANGLICISTE (ENSEIGNANT) [..., SPECIALITE : {angliciste}, ...]*
...

et l'agrégat EQUIPE/ENSEIGNANT [CDEQP : {mathématicien, informaticien, angliciste,}... , ...]

Naturellement, cette remarque n'a un sens que si la partition de $\mathcal{E}$ peut être décrite par une seule propriété support.

Représentation graphique des agrégats

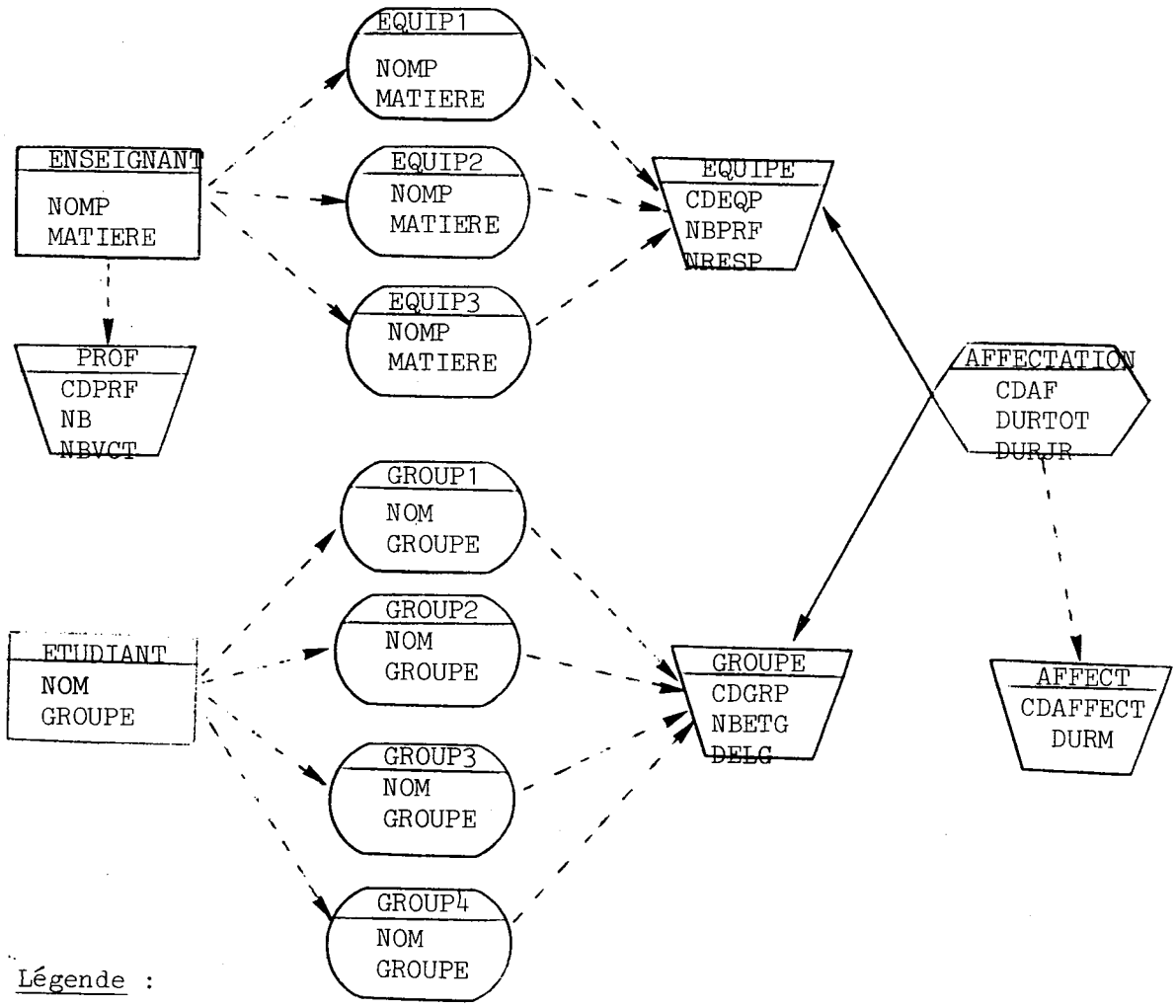
Symbolisme :



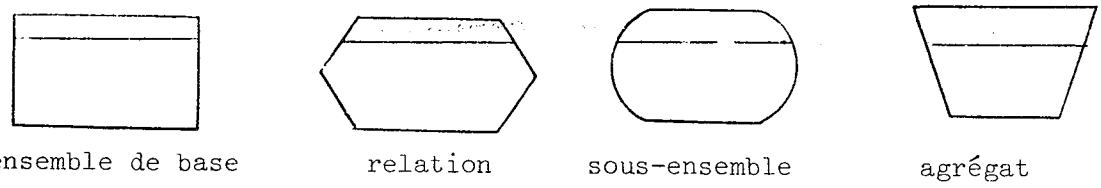
a : CODE d'un agrégat

b : CODE de la propriété-clé d'un agrégat + éventuellement les CODES d'autres propriétés.

Soit la représentation graphique relative à l'exemple 4.9.



Légende :



II.4.2 - Extensions du modèle des transactions

II.4.2.1 - Contrôles dans une transaction d'entrée

Nous distinguons deux types de contrôles sur les champs d'une transaction :

- les CONTROLES STANDARDS,
- les CONTROLES PARTICULIERS.

Contrôles standards d'un champ

Nous appelons CONTROLES STANDARDS ceux dont l'existence est obligatoirement déduite de la connaissance :

a) du TYPE de l'espace des valeurs de la PROPRIETE associée au champ,
b) ou du TYPE de la TRANSACTION.

TYPE d'une propriété :

Dans les spécifications d'une transaction d'entrée T, un champ du schéma de T est décrit par le CODE d'une propriété P (d'espace de valeurs V) de la structure logique des données et dans une REALISATION RT (exécution) de T, un champ est constitué des deux parties "CODE" et "VALEUR".

Lors de la lecture d'une réalisation RT, chaque champ est soumis à un CONTROLE STANDARD :
- de reconnaissance du CODE de la propriété P comme mot réservé obligatoire,
- de vérification de l'appartenance de la VALEUR à l'espace V.

Si ces deux actions sont sans erreurs, le champ est valide.

Exemple 4.12 :

Soient - des champs C1 : NOMP → ... →
C2 : TYPE → ... →
- des réalisations C1 : NOMP → dupont →
C2 : TYPE → cours →
- et les vérifications standards associées :

* *sur C1 : contrôler que "dupont" est une valeur de la propriété NOMP de type MOT, valeur de 30 caractères au plus*
* *sur C2 : "cours" est une valeur appartenant à la LISTE de MOTS définissant l'espace des valeurs de la propriété TYPE.*

Le tableau suivant résume les contrôles standards exécutés conformément aux types des propriétés (cf. § II.4.1.1).

Type de la propriété	CONTROLES STANDARDS de la valeur d'un champ
CODE	Il faut vérifier que la VALEUR fait au maximum 15 caractères et que cette valeur est discriminante par rapport à d'autres.
MOT, LIGNE, TEXTE	Vérification de la longueur de la VALEUR : 30 ou 60 caractères, ou 100 fois 60 caractères au plus.
NUMERIQUE BORNE ENTIER ou REEL	Le système vérifie que la VALEUR appartient bien à l'un des intervalles décrits pour l'espace des valeurs de la propriété.
LISTE de VALEURS - CODES - MOTS - ENTIERES - REELLES	La valeur doit appartenir à la liste décrite pour la propriété concernée.

TYPE d'une transaction :

Au cours de la description d'une transaction d'entrée T, il faut définir son TYPE (cf. § II.3.3.2), c'est-à-dire la nature de l'opération envisagée (déclaration, description, création, citation, modification ou suppression) sur l'élément objet de T.

Ce type est en fait la définition de CONTROLES STANDARDS des réalisations de T qui :

- **vérifient** que la valeur du premier champ est une valeur-clé d'un élément objet qui existe réellement dans le fichier ou la base de données, dans le cas d'une CITATION, DESCRIPTION, MODIFICATION ou SUPPRESSION.
- contrôlent que "le chemin de déclaration" (cf. § II.3.3.2) d'un élément est complet pour une DECLARATION ou CREATION.

Exemple 4.13 :

Pour la transaction TECREPROF,

NSECSOC → ... → NOMP → ... → PRNP → ... → ADRESSE2 → ... →

chemin de déclaration d'un enseignant

il faut vérifier la saisie des trois valeurs obligatoires (NSECSOC, NOMP, PRNP) ainsi que la valeur du champ NSECSOC qui doit être une valeur-clé discriminante d'un élément qui n'existe encore pas dans le système d'informations.

Contrôles particuliers

Ce sont tous les contrôles non définis automatiquement par les types des transactions ou les types des propriétés des champs.

Ils peuvent décrire :
- une signification particulière de la propriété associée à un champ ;
- des contrôles qui mettent en jeu plusieurs champs d'une même transaction ;
- des contraintes sémantiques sur les opérations secondaires définies sur des éléments liés à l'élément objet de la transaction.

Exemple 4.14 :

Reprenons la transaction TEMODENS :

NSECSOC → ... → CODENS → ... → CDMATIERE → ... → NIVEAU → ... → TYPE → ...

dont l'ensemble de référence est :

ENSEIGNANT [NSECSOC, ..., GRADE : {P, MC, MA, A, V}, ...]

et les ensembles concernés :

MATIERE [CDMATIERE, ...]

ENSEIGNEMENT (ENSEIGNANT, MATIERE) [CDENS, NIVEAU : {1,2,AS}, TYPE : {cours, td, tp}, ...]

On peut définir les contrôles particuliers :

P1 : contrôle attaché au champ TYPE pour vérifier que dans le cas d'une transaction portant sur un enseignant de grade P (professeur) ou MC (maître de conférence), l'enseignement décrit ne peut être que du type = cours.

P2 : contrôle attaché au champ NIVEAU, qui réalise un contrôle entre les champs NIVEAU et CDMATIERE, pour vérifier que c'est l'enseignement d'une matière prévue au programme pédagogique de l'année d'étude concernée (1 = 1ère année, 2 = 2ème année, AS = année spéciale).

P3 : sur le champ CODENS, pour indiquer que l'on désire déclarer un nouvel enseignement.

P4 : sur le champ CDMATIERE, pour préciser que l'on CITE une matière déjà déclarée ou créée.

P1 : précise une signification particulière de la propriété TYPE pour le champ.

P3 et P4 : indiquent des natures d'opérations relatives à la dynamique de gestion des éléments d'un système d'informations.

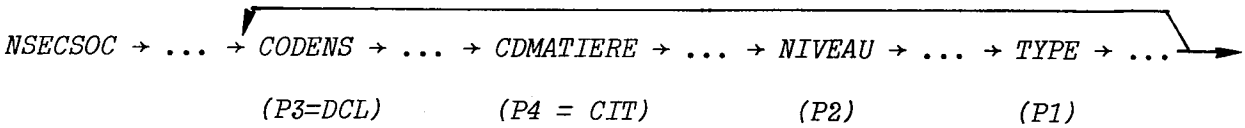
P2 : est un contrôle inter-champs.

Conventions nouvelles d'écriture du schéma d'une transaction d'entrée

Nous devons faire figurer dans le schéma tous les contrôles non standards dont l'importance est évidente ; nous citerons leurs noms, entre parenthèses, sous les champs, et nous indiquerons, s'il y a lieu, l'opération envisagée (DCL = déclarer, DEC = décrire, CIT = citer, CRE = créer, MOD = modifier, SUP = supprimer).

Exemple 4.15 :

Soit le schéma de la transaction TEMODENS :



Un texte de DEFINITION devra préciser le contenu d'un contrôle.

II.4.2.2 - Programme de calcul dans une transaction de sortie

En général, les valeurs des propriétés d'un élément d'un ensemble font l'objet de lectures dans des transactions d'entrées ; mais il en existe qui ne sont jamais lues et qui doivent être calculées [cf. § II.3.5].

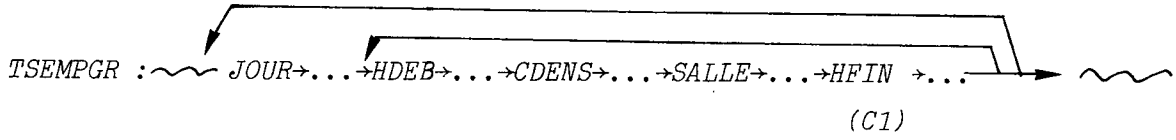
C'est le cas fréquent de l'AGE d'une personne. Des transactions d'entrée lisent la DATE DE NAISSANCE d'une personne, d'autres la DATE du JOUR ; l'AGE n'est jamais lu, mais appartient cependant à la liste des propriétés de l'ensemble PERSONNE. Si l'AGE figure dans une transaction de sortie sur une personne, il doit être préalablement calculé.

AGE est une propriété calculée de l'ensemble PERSONNE.

Au cours de la description du schéma d'une transaction de sortie, il est possible d'indiquer qu'un champ est défini par une propriété calculée ; il faut alors associer à ce champ le PROGRAMME de CALCUL correspondant. Nous citerons le nom d'un programme de calcul d'un champ entre parenthèses et nous indiquerons par ailleurs un texte de DEFINITION de ce programme.

Exemple 4.16 :

Décrivons le schéma d'une transaction de sortie TSEMPGR qui édite l'emploi du temps d'un groupe d'étudiants pour une semaine.



L'emploi du temps peut être vu comme une liste d'enseignements (CDENS) se déroulant dans des SALLES à partir d'un heure précise (HDEB) un JOUR donné. Un enseignement dure un certain temps (DUREE).

C1 = est un programme de calcul de l'heure de fin (HFIN) d'un cours, connaissant son heure de début (HDEB) et sa DUREE.

II.4.2.3 - Prologue et épilogue d'une transaction

Un PROLOGUE (ou EPILOGUE) d'une transaction T est un programme qui précède (ou suit) immédiatement et sans interruption une réalisation de T.

Il faut donner pour ces programmes :

- un NOM,
- un texte de DEFINITION de leur logique.

Exemple 4.17 :

Transactions d'entrée ou de sortie	*Prologues ou épilogues*
- *inscrire un étudiant*	*épilogue : enregistrer avec le dossier d'inscription la date du jour et le nom de la personne qui a réalisé cette inscription*
- *entrée de modifications d'un emploi du temps*	*épilogue : recalcul de l'emploi du temps modifié*
- *édition d'un document*	*prologue : édition d'un titre*
- *édition partielle d'un emploi du temps*	*prologue : recherche et construction du sous-emploi du temps concerné*

Remarque

Il peut exister a priori dans un système d'informations des traitements qui, à partir de données saisies, en élaborent d'autres.

Il faut se poser la question de savoir quel est le choix le plus judicieux de représentation d'un tel traitement :

- soit comme épilogue d'une transaction d'entrée,
- soit comme prologue d'une transaction de sortie.

Dans le premier cas, on considère au niveau fonctionnel que le traitement est une conséquence inévitable d'une opération de saisie à laquelle il succède.

Dans le deuxième cas, on considère que ce traitement n'a de raison d'être activé que pour les résultats qu'il produit dans le cadre de la transaction de sortie.

Règle méthodologique

Il est conseillé de faire figurer le traitement (prologue ou épilogue) auprès de la transaction où l'on peut dire que si elle est supprimée le traitement doit être aussi supprimé.

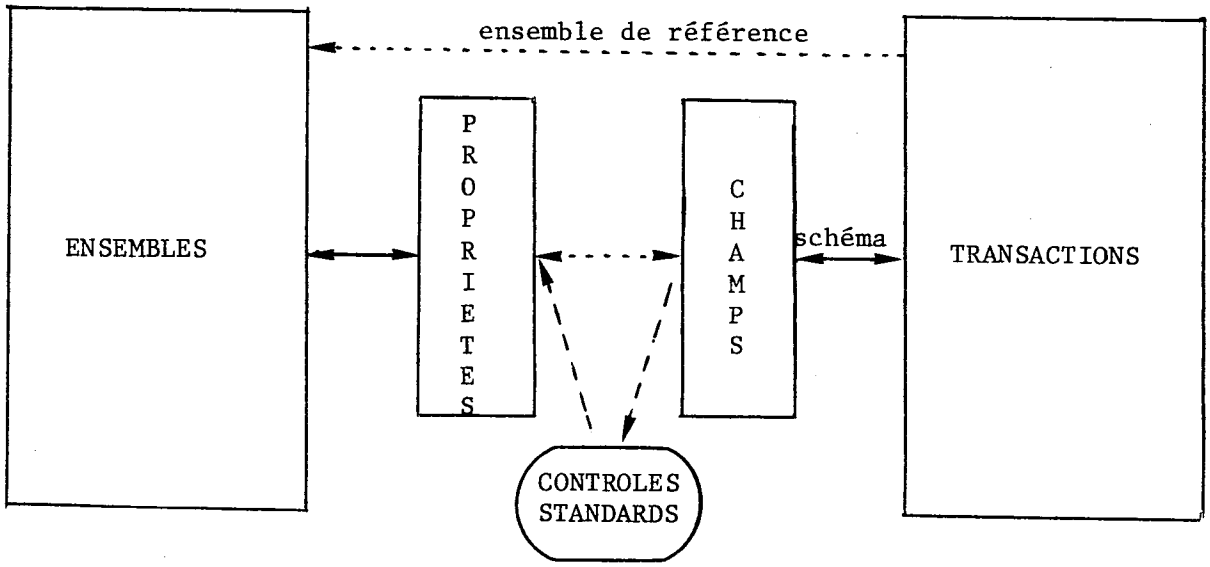
II-5 - CONCLUSION

II.5.1 - Les constituants de base du modèle et leurs interdépendances

MACSI-P propose une méthode rigoureuse de description fonctionnelle des données et des transactions d'une application informatique de gestion. Cette méthode est basée essentiellement sur les spécifications d'ENSEMBLES d'éléments et de TRANSACTIONS décrivant des opérations de déclaration, description, citation, création, modification, suppression ou édition d'éléments.

Ce modèle peut se résumer schématiquement par les trois figures suivantes.

Figure 1 - Le MODELE FONCTIONNEL D'APPLICATION.



"ENSEMBLE" étant le terme général utilisé dans le modèle pour définir des regroupements d'éléments qui sont des ensembles de base, des relations, des sous-ensembles ou des agrégats. De nombreuses combinaisons entre constituants sont autorisées ; elles augmentent les possibilités du modèle.

Ainsi, on peut décrire (cf. figure 2) :

- des ENSEMBLES de BASE,
- des RELATIONS entre n ensembles de base (e1),
- des SOUS-ENSEMBLES d'ensembles de base (e2),

- des RELATIONS entre relations (e3),
- des SOUS-ENSEMBLES de relations (e4),
- des RELATIONS entre sous-ensembles (e5),
- des RELATIONS entre ensembles de base et/ou relations et/ou sous-ensembles (e1, e3, e5),
- des SOUS-ENSEMBLES de sous-ensembles (e6),
- des AGREGATS d'ensemble de base (e7),
- des AGREGATS de relation (e8),
- des AGREGATS de sous-ensembles (e9),
- des RELATIONS entre agrégats (e10).

L'ensemble de référence d'une transaction est :

- soit un ensemble de base,
- soit une relation,
- soit un agrégat,
- soit un sous-ensemble.

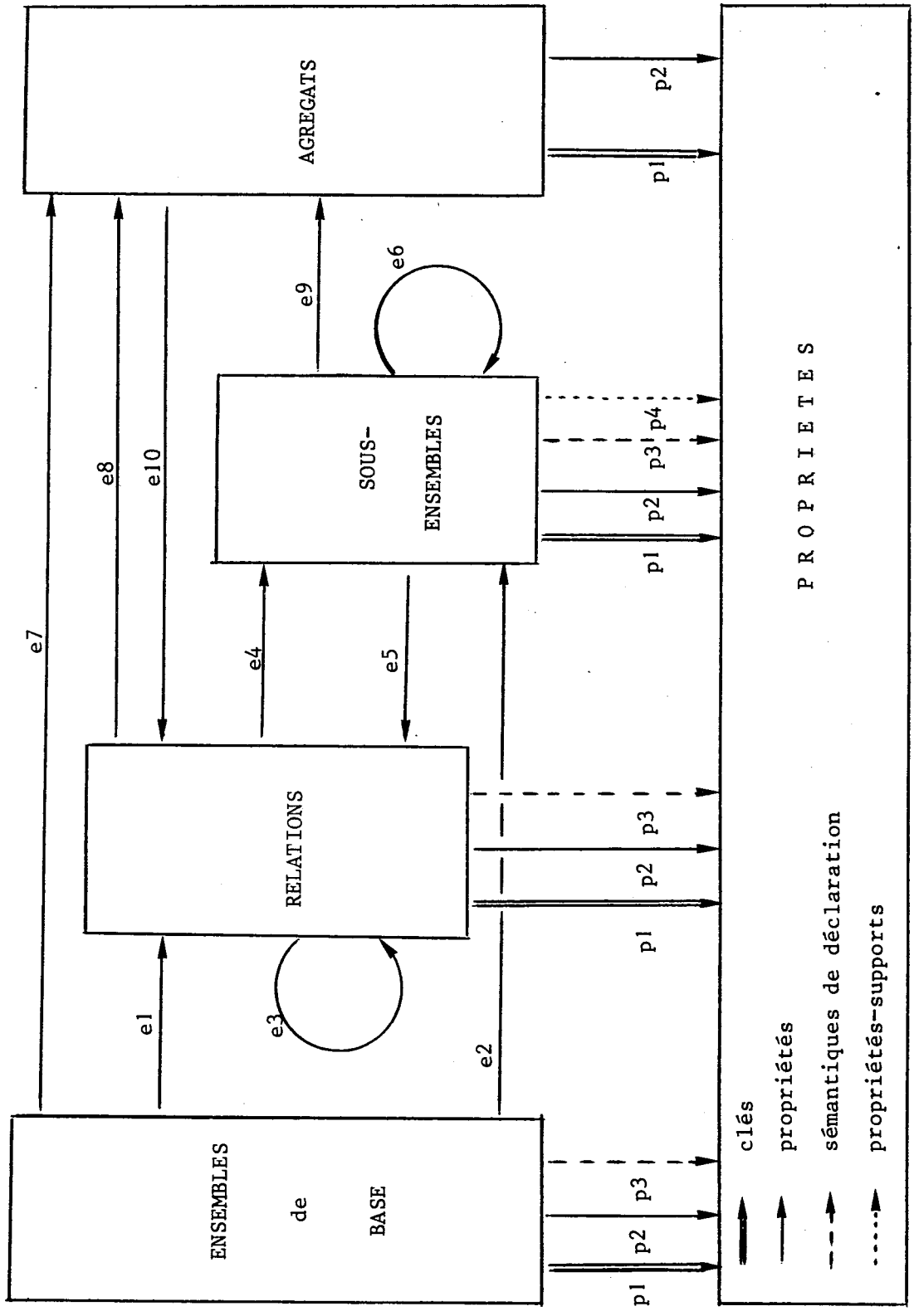


Figure 2 - Le MODELE de DESCRIPTION des DONNEES

Dans le modèle proposé, toute opération d'entrée, de sortie ou d'entrée-sortie est décrite par une TRANSACTION (cf. figure 3) et un SCHEMA de transaction (t1). Un schéma de transaction traduit le déroulement logique de l'OPERATION décrite (t3) en indiquant un enchaînement de CHAMPS (t2) à lire ou à écrire.

Des programmes spécifiques peuvent être associés aux schémas ou aux champs de transactions :

- programme assurant un CONTROLE PARTICULIER (t4) d'un champ en lecture ;
- programme de CALCUL (t5) interne de la valeur d'un champ en édition ;
- programmes qui doivent s'exécuter avant (PROLOGUE) (t6) ou après (EPILOGUE) (t7) une réalisation d'une transaction.

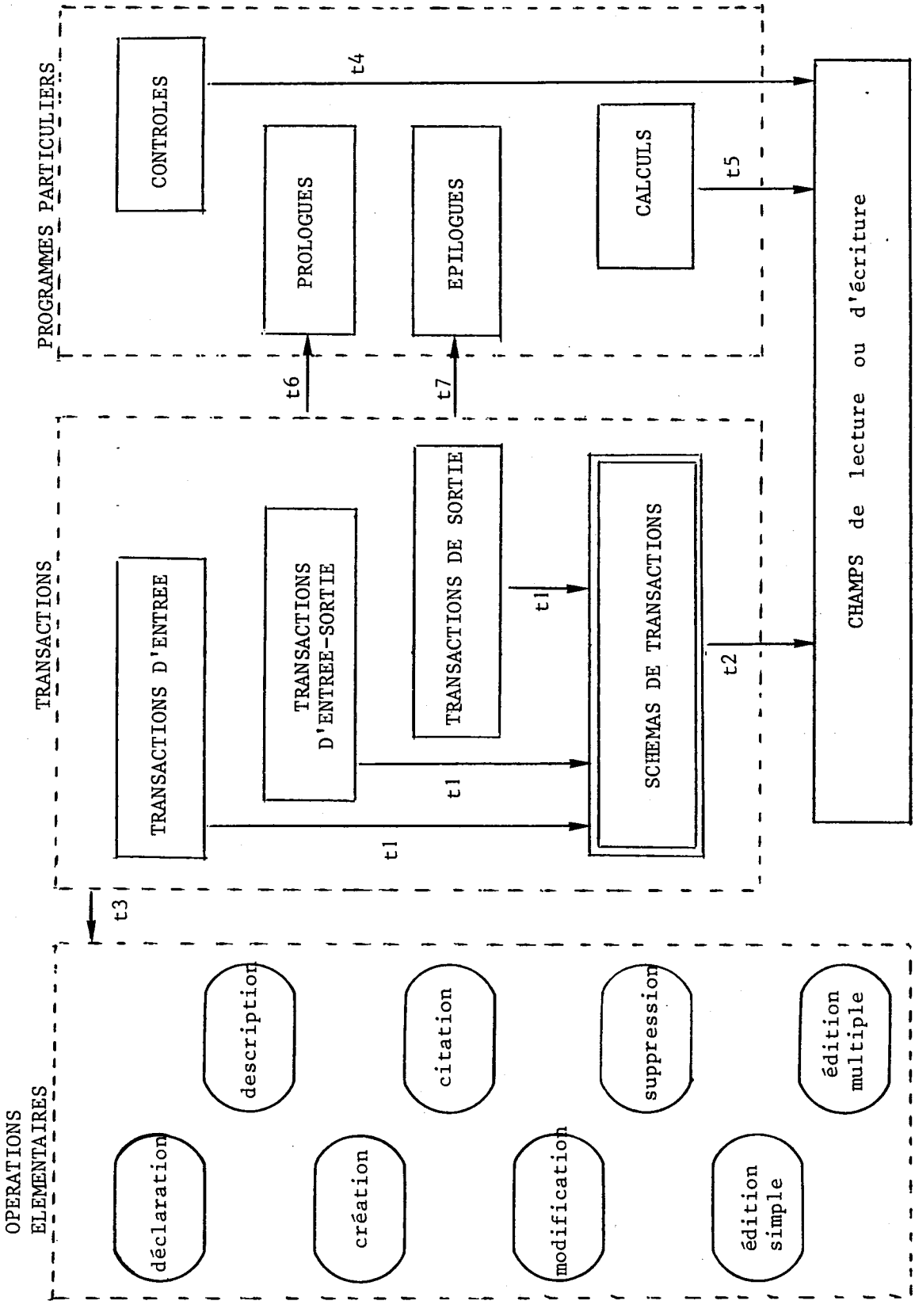


Figure 3 - Le MODELE de DESCRIPTION des TRANSACTIONS

Ce modèle peut être utilisé pour tenir à jour des inventaires afin de connaître :

- pour chaque transaction :
  - . la liste des propriétés citées,
  - . la liste des ensembles concernés directement (ensemble de référence), ou indirectement (relations ou arguments de relations avec l'ensemble de référence),
  - . la liste des programmes internes particuliers.

- pour chaque ensemble :
  - . la liste des transactions d'entrée,
  - . la liste des transactions de sortie,
  - . la liste des transactions d'entrée-sortie,
  - . la liste des opérations élémentaires définies.

- pour chaque propriété :
  - . la liste des transactions.

Ces listes permettent d'évaluer les conséquences d'une modification dans les spécifications d'un constituant sur d'autres constituants. Par exemple : quelles sont les transactions à modifier après la suppression d'une propriété d'un ensemble ?

II.5.2 - Récapitulatif des spécifications des constituants du modèle

Arrivé à ce point, nous avons vu l'ensemble des caractéristiques du modèle MACSI-P qui doivent être spécifiées pour que l'analyse fonctionnelle des données et des transactions d'une application informatique soit complète.

Nous pouvons résumer ces spécifications pour faciliter la lecture des chapitres suivants.

MODELE FONCTIONNEL	ENSEMBLES de BASE	RELATIONS	SOUS-ENSEM.	AGREGATS	PROPRIETES	TRANSACTIONS	CHAMPS
DEFINITION	ensemble d'éléments concrets	ensemble de liaisons entre éléments d'ensembles	ensemble partie d'un ensemble	ensemble d'ensembles	propriétés attachées aux éléments d'un ensemble	opérations automatisées d'échange d'informations	zone élémentaire d'un schéma de transaction
SPECIFICATIONS FONDAMENTALES et STRUCTURELLES	- CODE - 1 ou plusieurs CLES - liste de PROPRIETES - SEMANTIQUE de déclaration	- CODE - DEGRE - Liste des ARGUMENTS . FCT-DEPENDANCE . MIN . MAX - 1 ou plusieurs CLES - liste de PROPRIETES - SD	- CODE - ENSEMBLE-PERE - PROPRIETES-SUPPORTS - 1 ou plusieurs CLES - liste de PROPRIETES - SD	- CODE - ENSEMBLE de REFERENCE - Liste des ELEMENTS - 1 ou plusieurs CLES - liste de PROPRIETES /	- CODE - TYPE - COMPOSITION éventuelle - MODE	- CODE - SENS (E,S, E/S) - ENSEMBLE de REFERENCE - TYPE-OPERATION (dcl, dec, cre, cit, mod, sup, edt simple ou multiple) - SCHEMA - PROLOGUE - EPILOGUE	- CODE d'une propriété - un SENS : lecture ou écriture - des PROGRAMMES . contrôle standard . contrôle particulier . calcul ....
SPECIFICATIONS QUALITATIVES et DOCUMENTAIRES	- Libellé - DEFINITION - TAILLE	- Libellé - DEFINITION - TAILLE	- Libellé - DEFINITION - TAILLE	- Libellé - DEFINITION	- Libellé - DEFINITION	- Libellé - DEFINITION	- Libellé de la propriété - DEFINITION de la propriété

Figure 4 - Spécifications des constituants du modèle

Il nous reste à préciser maintenant deux autres aspects essentiels de la méthode :

a) Sous quelles formes et selon quelles règles faut-il rédiger les spécifications d'un projet ? C'est l'objet du chapitre III suivant.

b) Quelles sont les outils proposés et leurs possibilités d'utilisation ? Nous verrons l'ensemble des outils de MACSI-P au chapitre IV.

Chapitre III

Un Langage de Description

III-1 - Principes généraux du langage

III-2 - Instructions de spécification d'un modèle fonctionnel des données d'une application

III-3 - Instructions de spécification de transactions

III-4 - Instructions de mise au point et d'évaluation des spécifications.

III-1 - PRINCIPES GENERAUX DU LANGAGE

Le langage de description permet d'élaborer la *structure fonctionnelle d'une application* de gestion, ainsi que sa *documentation* par une suite de spécifications des différents composants qui la constituent (nous entendons par composant : un ensemble de base, une relation, un sous-ensemble, un agrégat, une propriété, une transaction d'entrée ou une transaction de sortie).

III-1-1 - Critères de choix d'un formalisme de définition logique des données et des traitements

Pour rédiger les spécifications d'une application, les "instructions" devaient satisfaire à de nombreux objectifs :

- Le langage doit permettre un meilleur dialogue utilisateur-concepteur-informaticien.
- L'utilisation des instructions doit être facilement compréhensible pour ne pas alourdir le travail de formation de ceux qui s'en serviront.
- Les instructions doivent être aisément modifiables en cas d'extension ou de modification du modèle MACSI-P pour l'adapter à une plus grande classe d'applications.
- Une instruction doit assurer non seulement la définition structurelle et sémantique d'un composant mais aussi sa documentation.
- Une même méthodologie à tous les niveaux (MACSI-P, prototype, ...) : une instruction MACSI-P ne réalise qu'une seule opération ("création", "description", "modification" ou "suppression") sur un composant C et éventuellement des opérations de "déclaration" ou "suppression" de composants C' liés fonctionnellement à C.
- Les instructions doivent exprimer l'ordre dans lequel sont introduites les valeurs associées aux composants concernés.
- Les différentes instructions ne doivent pas s'utiliser dans n'importe quel ordre. En effet, dans MACSI-P, nous voulons introduire une méthodologie de spécification de certains composants : une PROPRIETE, par exemple, ne peut pas être spécifiée sans avoir décrit au préalable un ENSEMBLE de rattachement. Nous préciserons un enchaînement des instructions au § III-2-3.

Aussi, avons-nous retenu le formalisme présenté au § II-2-3 : à toute instruction de spécification, nous avons associé une TRANSACTION en partie décrite par son SCHEMA de TRANSACTION.

III-1-2 - Différents états des spécifications d'un composant et types d'instructions

Rappelons qu'un composant peut être dans l'état "DECLARE" ou "DECRIT" (cf. § II-3-2-5).

Pour le modèle MACSI-P :
Un composant est DECLARE en donnant son CODE et son LIBELLE pour permettre son identification et sa documentation élémentaire.
Notons que l'association CODE+LIBELLE constitue la SEMANTIQUE de DECLARATION de tout composant du modèle MACSI-P.

Un composant est DECRIT si l'on a donné toutes les caractéristiques d'un composant DECLARE :

- en valorisant les caractéristiques propres de ce composant
- en déclarant de nouveaux composants qui lui sont liés

et/ou - en désignant des composants déclarés qui lui sont liés.

Dans le langage de spécification MACSI-P, nous distinguons 4 types d'instructions :

Instruction de CREATION d'un composant qui fait passer ce composant successivement à l'état DECLARE puis DECRIT.

Exemple : une instruction de création d'un ensemble de base doit :

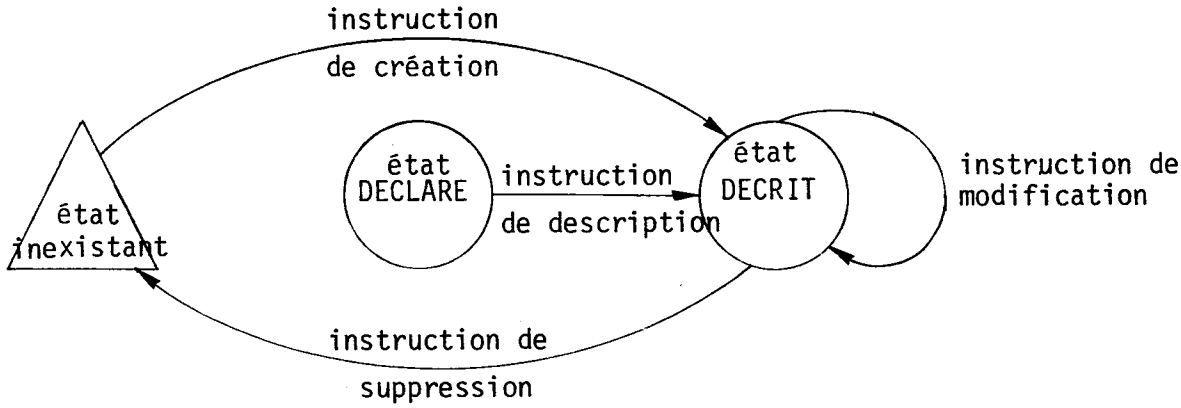
- déclarer cet ensemble, en donnant le CODE et le LIBELLE
- décrire cet ensemble, c'est-à-dire :
  - . donner la DEFINITION, la TAILLE
  - . déclarer les PROPRIETES de ses éléments
  - . désigner les PROPRIETES qui sont CLES
  - . désigner les PROPRIETES de la sémantique de déclaration de tout élément de l'ensemble.

Instruction de DESCRIPTION d'un composant DECLARE.

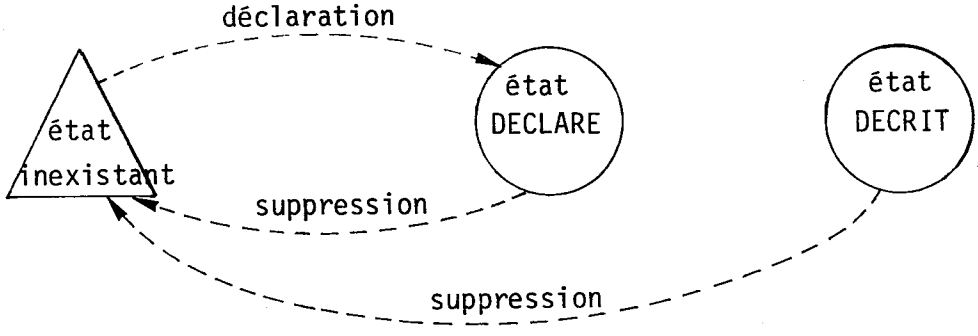
Instruction de MODIFICATION d'une ou plusieurs caractéristiques d'un composant DECRIT.

Instruction de SUPPRESSION de toutes les spécifications relatives à un composant considéré finalement comme inutile.

Nous pouvons résumer graphiquement les états des spécifications et les types des instructions qui permettent des transitions entre les états d'un composant.



Ces 4 instructions appliquées sur un composant peuvent contenir des opérations secondaires sur d'autres composants que nous représentons en pointillés sur le graphe suivant :



Exemple :

- *L'instruction de création (opération primaire) sur un composant C conduit ce composant à l'état DECRIT et peut faire passer des composants C' qui lui sont liés à l'état DECLARE (opération secondaire de déclaration).*
- *L'instruction de suppression (opération primaire) sur un composant C le fait passer à l'état INEXISTANT et peut conduire aussi des composants C' liés à C à l'état INEXISTANT (opération secondaire).*

Nous présentons successivement les instructions de spécification d'une structure logique de données d'une application (pour familiariser le lecteur avec le formalisme adopté) suivies des instructions pour spécifier des transactions d'entrée ou de sortie.

III-1-3 - Liste des instructions

Etant donné les constituants de base du modèle sémantique de MACSI-P; les noms des différentes instructions de spécification correspondant à une opération soit de création, soit de description, soit de modification, soit de suppression d'un composant du modèle sont résumés dans le tableau ci-dessous.

<table>
<tr><th>opération / composant</th><th>création</th><th>description</th><th>modification</th><th>suppression</th></tr>
<tr><td>ENSEMBLE de BASE</td><td>creens</td><td></td><td rowspan="4">modens</td><td rowspan="4">supens</td></tr>
<tr><td>RELATION</td><td>crerel</td><td></td></tr>
<tr><td>SOUS-ENSEMBLE</td><td>cresou</td><td></td></tr>
<tr><td>AGREGAT</td><td>creagr</td><td></td></tr>
<tr><td>PROPRIETE</td><td></td><td>decpp</td><td>modpp</td><td></td></tr>
<tr><td>TRANSACTION d'ENTREE</td><td>cretr-e</td><td></td><td rowspan="2">modtr</td><td rowspan="2">suptr</td></tr>
<tr><td>TRANSACTION de SORTIE</td><td>cretr-s</td><td></td></tr>
</table>

III-1-4 - Plan de présentation d'une instruction

Chaque instruction est définie systématiquement selon un plan de cinq rubriques :

1 - définition de l'*action* essentielle de l'instruction

2 - présentation graphique de sa *syntaxe* sous forme de SCHEMA de TRANSACTION

3 - description des conditions de *validité* de l'instruction et éventuellement des effets de bord qui l'accompagnent (contrôles STANDARDS et contrôles PARTICULIERS à une transaction)

4 - *utilisations* sur des exemples identiques à ceux choisis lors de la présentation du modèle au § II : ce sont des REALISATIONS de transactions

5 - liste des *messages d'erreurs* relatifs à l'aspect sémantique de l'instruction. Les messages d'erreurs sont de la forme :
ERREUR <numéro> SUR MOT : <chaîne> <libellé>
<numéro> est le code numérique de l'erreur ;
<chaîne> est la chaîne de caractère analysée ;
<libellé> est le texte explicatif de l'erreur.

D'autres messages d'erreurs peuvent être invoqués pour non respect du schéma de transaction : ce sont des erreurs dans la syntaxe de l'instruction ; ces messages sont décrits au § C-6-7.

Comme dans les pages précédentes, lorsque le terme "ensemble" est employé, il désigne indifféremment un ensemble de base, une relation, un sous-ensemble ou un agrégat.

III-2 - INSTRUCTIONS DE SPECIFICATION D'UN MODELE FONCTIONNEL DES DONNEES D'UNE APPLICATION

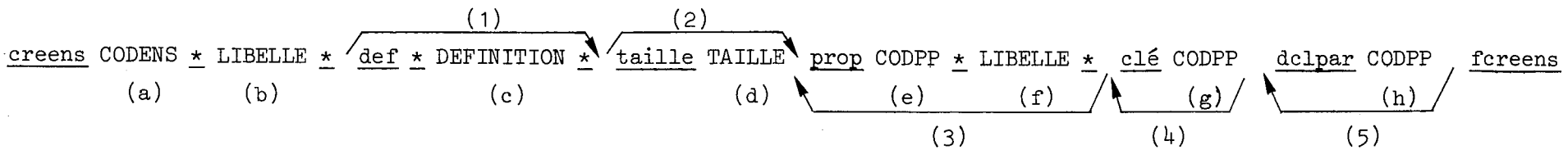
III-2-1 - Instructions de création et de description : CREENS, DECPP, CREREL, CRESOU, CREAGR

Ces cinq instructions permettent de spécifier respectivement des ensembles de base, des propriétés, des relations, des sous-ensembles et des agrégats constituant une structure logique de données d'une application.

III-2-1-1 - Instruction CREENS

action : créer un ensemble de base et déclarer des propriétés de ses éléments.

syntaxe :



validité :

Faisons d'abord des remarques qui s'appliquent à toutes les instructions du langage :

. une instruction commence et finit toujours par un mot clé (creens, fcreens, ...)

. les mots clés qui apparaissent dans une instruction sont toujours en lettres minuscules et soulignés (creens, *, def, taille, prop, ...) alors que les valeurs "variables" sont nommées en majuscules (CODENS, LIBELLE, DEFINITION, TAILLE, ...). Naturellement, cette convention n'est vraie que dans le cadre de cette présentation : sur un bordereau de perforation ou sur un terminal, on n'écrit ou ne frappe que des majuscules et on ne souligne rien.

. l'espace entre deux mots consécutifs (mots clés ou valeurs) est tout à fait laissé au choix de l'utilisateur.

. les valeurs de type "mot", "ligne" ou "texte" sont toujours encadrées par deux étoiles marquant le début et la fin du mot, de la ligne ou du texte.

. des lettres ou chiffres, entre deux parenthèses ((a), (b), ... (1), ...) servent de repères pour commenter une valeur ou un cheminement du graphe.

. la validité d'une instruction sera en partie décrite dans un tableau à 5 colonnes :

- les 2 premières colonnes indiquent le "nom" de la zone ou du chemin de l'instruction et le repère éventuel
- une colonne précise le type de la valeur à lire qui caractérise le "contrôle standard" à faire sur la zone :

TYPE =

1 = CODE (cf. § II-4-1-1)
2 = MOT
3 = ENTIER
4 = REEL
5 = Liste de CODES
6 = Liste de MOTS
7 = Liste d'ENTIERS
8 = Liste de REELS
9 = LIGNE
10 = TEXTE
0 = mot réservé (mots clés du langage MACSI-P)

- une 4ème colonne indique si la valeur à fournir pour ce champ par l'utilisateur de l'instruction, doit être recherchée parmi des valeurs déjà enregistrées pour ce modèle d'application (notée R : recherche d'une valeur déjà donnée) ou si c'est une valeur qui n'a encore jamais été saisie (notée N : nouvelle valeur à donner).
- la 5ème colonne regroupe différents commentaires précisant la sémantique de la zone, des "contrôles particuliers", des conseils et de bonnes utilisations pour clarifier l'écriture.
- on indiquera par un / dans les colonnes 1, 3 et 4, que la colonne 5 contient un commentaire sur un chemin repéré dans la colonne 2.

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
CODENS	(a)	1	N	- code que l'on veut attribuer à cet ensemble de base à créer - ce code doit être discriminant par rapport à tous les codes des composants déjà spécifiés
LIBELLE	(b)	9	N	- le LIBELLE assure la documentation élémentaire de l'ensemble de base - le CODENS est en général une abréviation de ce libellé.
DEFINITION	(c)	10	N	- DEFINITION de l'ensemble en extension ou en compréhension - le texte peut être réduit à une valeur particulière (ras ou u), si la définition est inutile (LIBELLE en (b) assez explicite), ou inconnue pour le moment.
/	(1)	/	/	- cette dernière possibilité (def * u *) est avantageusement remplacée par l'omission complète (1) du chemin def * DEFINITION *.
TAILLE	(d)	3	N	- valeur du cardinal maximum de l'ensemble (nombre maximum d'éléments de l'ensemble).
/	(2)	/	/	- le chemin taille TAILLE est facultatif (2)
CODPP	(e)	1	N	- code d'une nouvelle propriété à spécifier pour les éléments de l'ensemble - ce code doit être discriminant par rapport à tous les codes enregistrés.
LIBELLE	(f)	9	N	- documentation de la propriété identifiée par CODPP (e) - le chemin "prop CODPP * LIBELLE * assure la DECLARATION d'une propriété dans la structure logique.
/	(3)	/	/	- on répète ce chemin s'il y a encore des propriétés à déclarer pour les éléments de l'ensemble.
CODPP	(g)	1	R	- code d'une propriété clé de l'ensemble, la propriété a dû faire l'objet d'une déclaration préalable ((e) + (f)).
/	(4)	/	/	- cette itération permet d'attribuer plusieurs clés à un même ensemble.
CODPP	(h)	1	R	- code d'une propriété, déclarée auparavant en (e) et (f), pour décrire la sémantique de déclaration des éléments de l'ensemble.
/	(5)	/	/	- la répétition du chemin autorise la description d'une sémantique de déclaration constituée de plusieurs propriétés. - la première réalisation du chemin "dclpar CODPP" concerne obligatoirement le code d'une propriété désignée comme clé en (g).

Il faut remarquer que c'est seulement lorsque les propriétés d'un ensemble sont déclarées que l'on peut dire quelles en sont les clés et les constituants de la sémantique de déclaration.

Notons aussi qu'il est nécessaire de DECLARER au moins une PROPRIETE et d'attribuer au moins une CLE à un ensemble, ainsi que de définir une SEMANTIQUE de DECLARATION constituée d'au moins une PROPRIETE-CLE.

- effets de bord :

Dans le cas du choix des chemins (1) ou (2), les caractéristiques correspondantes DEFINITION ou TAILLE prennent la valeur u pour l'ensemble identifié par CODENS (a).

- utilisations : DESCRIPTION de l'ensemble ETUDIANT

```
creens ETUDIANT * étudiants de l'IUT *        } DECLARATION de l'ensemble ETUDIANT
       def * ensemble des étudiants inscrits à l'IUT2 de Grenoble avantle 15 Septembre de l'année en cours *
       taille 1000
       prop PRN * prénom *
       prop NOM * ras *
       prop NUMETUD * numéro d'étudiant *
       prop ADR * adresse personnelle de l'étudiant *
       prop NSECSOC * numéro de sécurité sociale *
       prop DEPT * département IUT d'inscription *
       prop ANNEE * cycle 1, 2 ou spécial *
                                              } DECLARATIONS de nouvelles propriétés
                                                (les valeurs fournies sont nouvelles : N en col. 4)
       clé NSECSOC                            } certaines propriétés déclarées précédemment sont des CLES de l'ensemble
       clé NUMETUD                              (codes déjà donnés : R en col. 4)
       dclpar NUMETUD                         } définition de la SEMANTIQUE de DECLARATION de l'ensemble ETUDIANT
       dclpar NOM                               (codes de propriétés déjà déclarées : R en col. 4)
       dclpar PRN
fcreens
```

La grille minimale de création d'un ensemble de base est : creens ... * ... * prop ... * ... * clé ... dclpar ... fcreens
comme par exemple : creens UV * unités de valeur * prop CODUV * code unité de valeur *
clé CODUV dclpar CODUV fcreens

Nous donnons encore d'autres exemples de créations d'ensembles de base qui sont utilisés dans les instructions suivantes pour créer des relations, des sous-ensembles, des agrégats ou des transactions.

```
creens FILIERE * cycle d'enseignement *
       taille 30
       prop CODFIL * code de la filière *
       prop LIBFIL * libellé de la filière *
       clé CODFIL
       dclpar CODFIL
fcreens

creens SALLE * salle de cours de l'IUT *
       taille 100
       prop CDSALLE * code de la salle *
       prop CAPACITE * nombre de places assises *
       prop CDBAT * code bâtiment *
       prop ETAGE * ras *
       clé CDSALLE
       dclpar CDSALLE
fcreens
```

creens BUREAU * bureau de professeurs *
taille 50
prop NUM-B * numéro du bureau *
prop NBPROF * nombre de professeurs dans le bureau *
clé NUM-B
dclpar NUM-B
fcreens

- messages d'erreurs : (numéros et libellés)

200 * IL EXISTE DEJA UN ENSEMBLE OU RELATION AYANT CE CODE *

201 * IL EXISTE DEJA UN OBJET DE NATURE DISTINCTE AYANT CE CODE *

202 * LA TAILLE DOIT ETRE COMPRISE ENTRE 1 ET 9999 *

203 * IL EXISTE DEJA UNE PROPRIETE AYANT CE CODE *

204 * LA CLE UTILISEE N'EST PAS PROPRIETE DE L'ENSEMBLE EN COURS *

205 * CETTE PROPRIETE EST DEJA CLE DE CET ENSEMBLE *

206 * CE DOIT ETRE LE CODE D'1 PROP DECLAREE DANS L'ENSEMBLE *

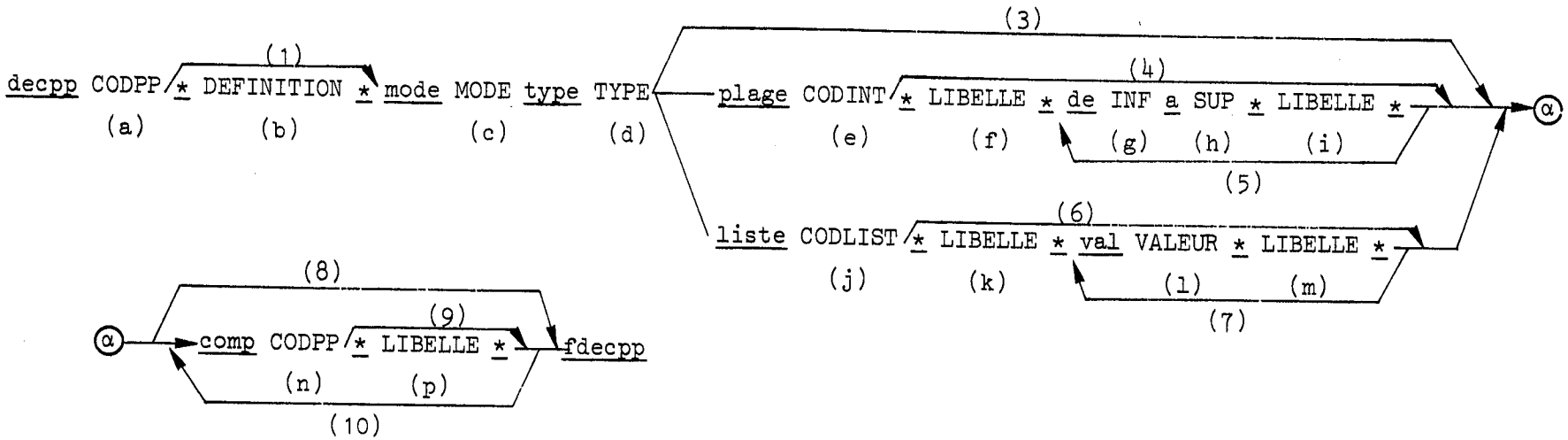
207 * 1er CODE DOIT ETRE CODE D'UNE PROP CLE DE L'ENSEMBLE *

208 * CODE D'UNE PROP DEJA INDIQUEE DANS LA SEM DE DECLARATION *

III-2-1-2 - Instruction DECPP

action : décrire une propriété déclarée lors d'une instruction de création ou de modification d'un ensemble (ensemble de base, relation, sous-ensemble ou agrégat), ou une instruction de description ou modification d'une propriété composée.

syntaxe :



validité :

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
CODPP	(a)	1	R	- code d'une propriété déjà DECLAREE et non encore DECRITE dans la structure logique de données.
DEFINITION	(b)	10	N	- texte donnant la signification exacte de la propriété et les contrôles éventuels à prévoir.
/	(1)	/	/	- ce chemin est utilisé de préférence à "def * u *" si l'on ne désire pas définir immédiatement la propriété.
MODE	(c)	7	N	- mode d'élaboration de la valeur de cette propriété pour un élément - MODE = 1 : valeur saisie MODE = 2 : valeur calculée.
TYPE	(d)	7	N	- type de l'espace de données associé à la propriété - ce type doit appartenir à une liste rappelée ici : = 1 : code = 3 : entier = 5 : liste de codes = 7 : liste d'entiers = 9 : ligne = 2 : mot = 4 : réel = 6 : liste de mots = 8 : liste de réels = 10 : texte - si la propriété est CLE, elle doit être obligatoirement de type = 1 ou 5.
/	(3)	/	/	- chemin pris si TYPE (d) = 1, 2, 9 ou 10
plage	/	0	/	- chemin pris si TYPE (d) = 3 ou 4
CODINT	(e)	1	R,N	- code d'un intervalle ou d'un ensemble d'intervalles de définition de la propriété
/	(4)	/	/	- on prend ce chemin si cet intervalle (ou cet ensemble d'intervalles) a déjà été créé, sinon il faut continuer à cheminer dans le graphe pour le créer.
LIBELLE	(f)	9	N	- libellé du nouvel ensemble d'intervalles CODINT (e)
INF	(g)	3 ou 4	N	- pour décrire la borne inférieure, la borne supérieure et le libellé d'un intervalle fermé de CODINT (e)
SUP	(h)	3 ou 4	N	
LIBELLE	(i)	9	N	- INF et SUP de type 3 si TYPE (d) = 3 - INF et SUP de type 4 si TYPE (d) = 4
/	(5)	/	/	- ce chemin est utilisé pour décrire plusieurs intervalles du même ensemble CODINT (e)
liste	/	0	/	- ce chemin est pris si TYPE (d) = 5, 6, 7 ou 8
CODLIST	(j)	1	R,N	- code d'une liste de valeurs définissant une propriété

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
/	(6)	/	/	- chemin emprunté si la liste de code CODLIST (j) a déjà été créée, sinon on continue.
LIBELLE	(k)	9	N	- libellé de la nouvelle liste CODLIST (j) à créer
VALEUR	(l)	1,2,3 ou 4	N	- description et documentation d'une valeur de la liste CODLIST (j)
LIBELLE	(m)	9	N	VALEUR doit être de type = 1 si TYPE (d) = 5 " " " 2 " 6 " " " 3 " 7 " " " 4 " 8
/	(7)	/	/	- itération pour décrire chaque valeur de la liste CODLIST (j)
CODPP	(n)	1	R,N	- code d'une propriété composante de la propriété composée CODPP (a) - si c'est le code d'une propriété déjà déclarée dans la structure logique, alors cette propriété doit obligatoirement appartenir au même ensemble que la propriété CODPP (a)
/	(9)	/	/	- ce chemin est utilisé si la propriété CODPP (n) a déjà été déclarée
LIBELLE	(p)	9	N	- documentation de CODPP (n) dans le cas de la déclaration d'une nouvelle propriété
/	(10)	/	/	- pour décrire toutes les composantes d'une propriété composée CODPP (a)
/	(8)	/	/	- chemin utilisé si CODPP (a) est une propriété élémentaire

- utilisations: decpp NOM mode 1 type 2 fdecpp

decpp NSECSOC * code 13 caractères numériques en 6 zones *

mode 1 type 1

comp SEXE * ras *

comp AN * année de naissance *

comp MOIS * mois de naissance *

comp DPT * département de naissance *

comp COM * commune de naissance *

comp NUM * numéro d'ordre de naissance *

fdecpp

decpp ANNEE * un étudiant de l'IUT est en 1ère, 2ème année ou année spéciale *

mode 1

type 6 liste L47 * liste des cycles de l'IUT * val A1 * 1ère année *
val A2 * 2ème année *
val AS * année spéciale *

fdecpp

decpp CAPACITE * nombre de places assises *

mode 1

type 3 plage I63 * capacité d'une salle * de 1 à 500 * ras *

fdecpp

decpp DEPT * ras *

mode 1

type 6 liste L48 * liste des départements de l'IUT * val INFO * informatique *
val GEA * gestion des entreprises et administrations *
val TCO * techniques de commercialisation *
val CJJ * carrières juridiques et judiciaires *
val STQ * statistiques, études économiques et techniques quantitatives de gestion *
val SOC * carrières sociales *

fdecpp

- messages d'erreurs (numéros et libellés)

220 * CE CODE IDENTIFIE UN OBJET QUI N'EST PAS UNE PROPRIETE *

221 * OBJET NON EXISTANT *

222 * LE MOT CLE PLAGE SUIT OBLIGATOIREMENT TYPE = 3 ou 4 *

223 * LE MOT CLE LIST SUIT OBLIGATOIREMENT TYPE = 5, 6, 7 ou 8 *

224 * LE MOT CLE COMP OU FDECPP SUIT OBLIGATOIREMENT TYPE = 1, 2, 9, 10 *

225 * CODE NOUVEAU, LE LIBELLE DOIT SUIVRE *

226 * CODE EXISTANT : PAS DE LIBELLE A FOURNIR *

227 * EXISTE 1 OBJET (NON INTERVALLE) DE NAT. DISTINCTE AYANT CE CODE *

228 * EXISTE 1 OBJET (NON LISTE VAL.) DE NATURE DISTINCTE AYANT CE CODE *

229 * POUR UN INTERVALLE : 0 <BORNE INF < BORNE SUP < 999999999 *

230 * IL FAUT BORNE SUP > BORNE INF *

231 * VALEUR NON NUMERIQUE *

232 * CETTE VALEUR EXISTE DEJA DANS LA LISTE *

233 * UNE PROPRIETE CLE COMPOSEE DOIT ETRE DE TYPE = CODE *

234 * LIBELLE OBLIGATOIRE DANS LE CAS D'UNE NOUVELLE PROPRIETE *

235 * PROPRIETE COMPOSANTE DEJA DONNEE *

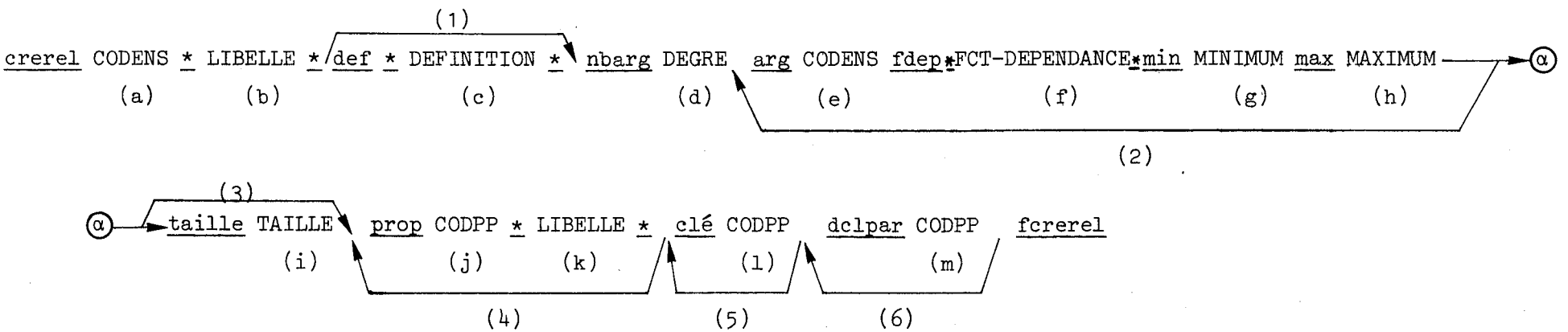
236 * PROPRIETE COMPOSANTE DOIT APPARTENIR AU MEME ENSEMBLE QUE PROPRIETE COMPOSEE *

237 * PROPRIETE DEJA EXISTANTE, DONC LIBELLE INTERDIT *

III-2-1-3 - Instruction CREREL

action : créer une relation binaire ou n-aire entre des ensembles (de base, relations, sous-ensembles ou agrégats) et déclarer des propriétés de ses éléments.

syntaxe :



validité :

Zone	Repère	Type	N/R	COMMENTAIRES ET CONTROLES PARTICULIERS
CODENS	(a)	1	N	- code que l'on veut attribuer à la relation à créer (une relation est un ensemble particuli
				- ce code doit être discriminant par rapport à tous les codes des composants déjà spécifiés.
LIBELLE	(b)	9	N	- pour expliquer brièvement CODENS (a)
DEFINITION	(c)	10	N	- DEFINITION précise et fonctionnelle de la relation à créer
/	(1)	/	/	- ce chemin rend cette définition facultative et peut remplacer def * u * par exemple
DEGRE	(d)	3	N	- le DEGRE d'une relation est le nombre d'ensembles-arguments de celle-ci
				- ce DEGRE doit être supérieur ou égal à 2
CODENS	(e)	1	R	- code d'un ensemble (de base, relation, sous-ensemble ou agrégat) déjà créé
				- ce code identifie un ensemble-argument de la relation CODENS (a)
FCT-DEPENDANCE	(f)	2	N	- nom de la fonction de dépendance relative à l'argument qui vientd'être identifié par CODENS (e)
				- les fonctions de dépendance d'une même relation doivent avoir des noms distincts
MINIMUM	(g)	3	N	- MINIMUM et MAXIMUM relatifs à la fonction de dépendance définie en (f)
MAXIMUM	(h)	3	N	- ce sont 2 entiers tels que MIN ≥ 0, MAX ≥ 1, MIN ≤ MAX et MAX ≤ TAILLE de l'ensemble identifié par CODENS (e)
/	(2)	/	/	- répétition du chemin "arg ... fdep ... min .., max ..." autant de fois ("DEGRE" fois) que la relation a d'arguments
TAILLE	(i)	3	N	- nombre maximum d'éléments de la relation CODENS (a) (nombre de n-uplets liens entre les arguments)
/	(3)	/	/	- le chemin "taille TAILLE" est facultatif
CODPP	(j)	1	N	- ce chemin prop CODPP * LIBELLE * assure la DECLARATION d'une nouvelle propriété des
LIBELLE	(k)	9	N	éléments de la relation CODENS (a)
				- le code doit être discriminant
/	(4)	/	/	- chaque répétition de ce chemin permet de déclarer une nouvelle propriété
CODPP	(l)	1	R	- code d'une propriété déclarée en (j) et qui est CLE pour la relation (a)
/	(5)	/	/	- ce chemin permet d'attribuer plusieurs clés à une relation
CODPP	(m)	1	R	- code d'une propriété déclarée en (j) et appartenant à la sémantique de déclaration des éléments de la relation (a), ou code d'une propriété clé d'un des arguments de la relation

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
/	(6)	/	/	- chemin pris si cette sémantique de déclaration est constituée de plusieurs propriétés. - le premier parcours du chemin "dclpar CODPP" concerne toujours le code d'une propriété désignée comme clé en (1) - On peut remarquer que les arguments, CODENS (e), doivent être des ensembles déjà créés par l'une des instructions creens, crerel, cresou ou creagr, et qu'il est nécessaire de déclarer au moins une propriété, de désigner au moins uneclé et de définir au moins un constituant (clé) de la sémantique de déclaration pour les éléments d'une relation.

Utilisations :

```
crerel INSCRPT1 * ensemble des couples (étudiant, unité de valeur) *
      def * un étudiant doit faire une inscription dans chaque unité de valeur choisie,
            un étudiant qui est déjà inscrit dans une filière ne doit pas faire d'inscriptions dans des unités de valeurs *
      nbarg 2
      arg ETUDIANT fdep*inscrits*min 10 max 40
      arg UV fdep*s'inscrit-dans*min 0 max 8
      taille u
      prop CODINS1 * code d'une inscription *
      prop NATURE * =1, cours du soir ; =2 mi-temps ; =3 plein-temps *
      prop CTL * modalité de contrôle des connaissances choisie *
      clé CODINS1
      dclpar CODINS1 dclpar NUMETUD dclpar CODUV
fcrerel
crerel INSCRPT2 * ensemble des couples (étudiant, filière) *
      nbarg 2
      arg ETUDIANT fdep*inscrits-à*min 50 max 300
      arg FILIERE fdep*s'inscrit-à*min 0 max 1
      prop CODINS2 * code d'une inscription dans une filière *
      clé CODINS2
      dclpar CODINS2 dclpar NUMETUD dclpar CODFIL
fcrerel
```

- messages d'erreurs (numéros et libellés) :

260 * IL FAUT CITER LE CODE D'1 ENSEMBLE, RELATION, AGREGAT, SOUS-ENSEMBLE *

261 * 0 <= MIN (FONCT.DEP) <= TAILLE DE L'ARGUMENT *

262 * MIN <= MAX (FONCT. DEP.) <= TAILLE DE L'ARGUMENT *

263 * DEGRE D'1 RELATION (NBARG) SUP A 10 OU INF A 2 *

264 * NOMBRE DESCRIPTIONS D'ARGUMENTS > DEGRE DE LA RELATION *

265 * 1er CODE DOIT ETRE CODE D'1 PROPRIETE CLE DE LA RELATION *

266 * MANQUE CODES DE PROPRIETE-CLE DES ARGUMENTS DANS SEM-DCL RELATION *

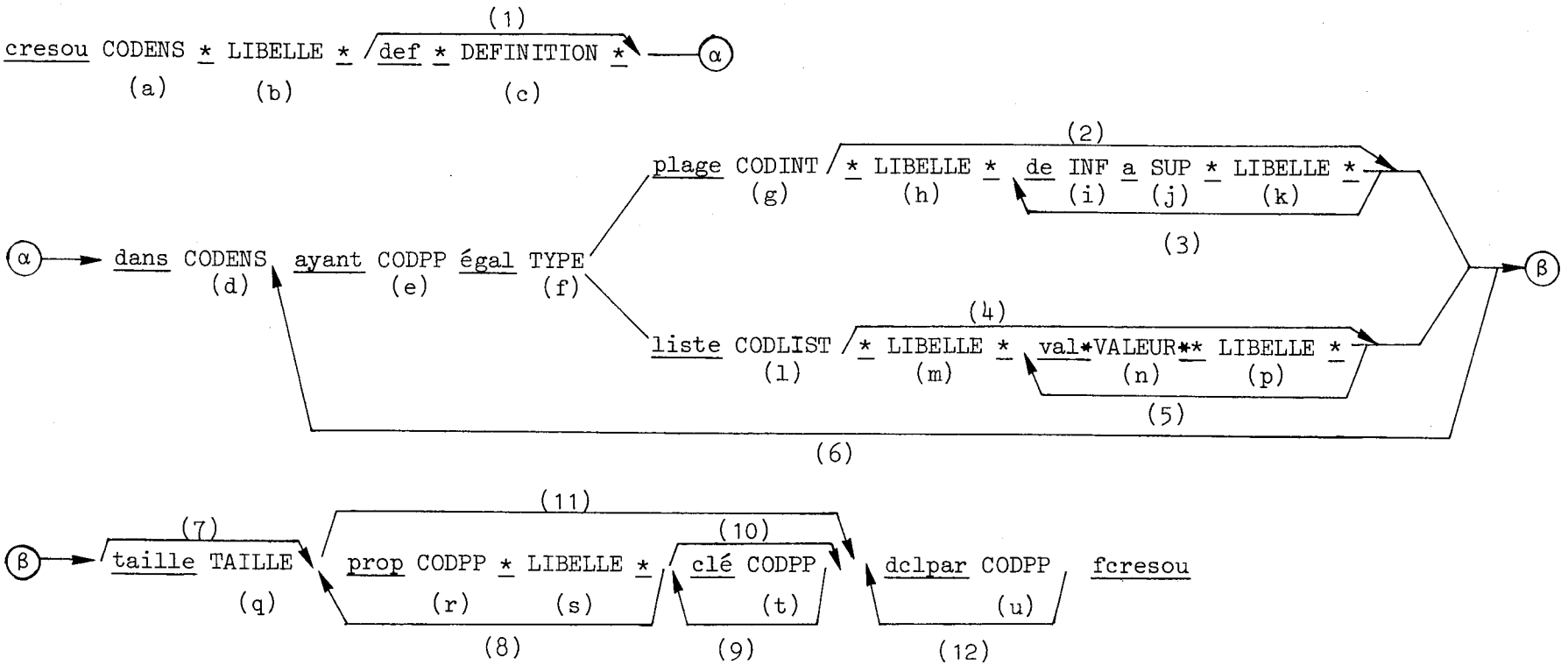
267 * DOIT ETRE CODE D'1 PROPRIETE DE LA RELATION *

268 * CODE DEJA DONNE DANS LA SEM DE DCL DE LA RELATION *

III-2-1-4 - Instruction CRESOU

action : créer un sous-ensemble d'un ensemble de base, d'une relation, ou d'un sous-ensemble, et déclarer éventuellement des propriétés propres à ses éléments.

syntaxe :



validité :

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
CODENS	(a)	1	N	- CODE et LIBELLE du sous-ensemble à créer (un sous-ensemble est un ensemble particulier)
LIBELLE	(b)	9	N	- CODENS doit être discriminant.
DEFINITION	(c)	10	N	- texte définissant en extension ou en compréhension le sous-ensemble
/	(1)	/	/	- cette DEFINITION est facultative.
CODENS	(d)	1	R	- code d'un ensemble de base, d'une relation ou d'un sous-ensemble déjà CREE, dont on veut CREER un sous-ensemble CODENS (a)
ayant	/	0	/	- chemin de description d'une propriété "support de la partition"
CODPP	(e)	1	R	- c'est le code d'une propriété "SUPPORT de la PARTITION"
				- ce code doit être celui d'une propriété déjà DECLAREE et attribuée à l'ensemble CODENS (d)
				- la propriété ainsi identifiée par CODPP (e) doit être de TYPE = 1,2,3,4,5,6,7 ou 8
TYPE	(f)	7	N	- TYPE précise l'espace de données associé à la propriété support CODPP (e)
				- TYPE (f) ne peut prendre que les valeurs 3, 4, 5, 6, 7 ou 8
				- TYPE (f) doit être compatible avec le TYPE défini pour la propriété CODPP (e) dans l'ensemble CODENS (d). Nous pouvons résumer cette compatibilité dans une table :

TYPE de la propriété dans l'ensemble CODENS (d)	1	2	3	4	5	6	7	8	9	10
TYPE (f) de la propriété dans le sous-ensemble CODENS (a)	5	6	3 ou 7	4 ou 8	5	6	7	8	impossible	

plage	/	0	/	- ce chemin "plage ....." est pris si et seulement si TYPE (f) = 3 ou 4
CODINT	(g)	1	R ou N	- code d'un intervalle ou d'un ensemble d'intervalles définissant l'espace de données de
LIBELLE	(h)	9	N	la propriété support CODPP (e)
/	(2)	/	/	- si cet ensemble d'intervalles a déjà fait l'objet d'une CREATION alors on ne donne pas son LIBELLE (h) et on emprunte le chemin (2)
				- pour créer un nouvel ensemble d'intervalles, CODINT doit être discriminant et on donne son LIBELLE (h) ainsi que la description de chacun de ses intervalles par INF (i), SUP (j) et LIBELLE (k).

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
INF	(i)	3 ou 4	N	- description d'un intervalle par sa borne inférieure (i), sa borne supérieure (j) et son libellé (k)
SUP	(j)	3 ou 4	N	
LIBELLE	(k)	9	N	- INF et SUP sont de type 3 si TYPE (f) = 3 et INF et SUP sont de type 4 si TYPE (f) = 4
/	(3)	/	/	- ce chemin permet de décrire chaque intervalle de l'ensemble d'intervalles CODINT (g)
liste	/	0	/	- ce chemin "liste ..." est pris si et seulement si TYPE (f) = 5, 6, 7 ou 8
CODLIST	(1)	1	R ou N	- code d'une liste de valeurs définissant la propriété support CODPP (e)
/	(4)	/	/	- si cette liste existe déjà, on emprunte ce chemin, sinon le code doit être discriminant et l'on doit décrire cette liste
LIBELLE	(m)	9	N	- documentation d'une nouvelle liste CODLIST (1)
VALEUR	(n)	1,2,3 ou 4	N	- description et documentation d'une valeur de la liste CODLIST (1)
LIBELLE	(p)	9	N	- VALEUR est de type 1 si TYPE (f) de la liste est = 5 2 = 6 3 = 7 4 = 8
/	(5)	/	/	- ce chemin permet de décrire chaque valeur de la liste
/	(6)	/	/	- ce chemin permet la description d'une autre propriété support de la partition - le nombre de propriétés supports de la partition n'est pas limité (si ce n'est par le nombre de propriétés de l'ensemble CODENS (d))
TAILLE	(q)	3	N	- nombre maximum d'éléments du sous-ensemble CODENS (a) - TAILLE doit être inférieure ou égale à la "taille" de l'ensemble CODENS (d)
/	(7)	/	/	- ce chemin "taille TAILLE" est facultatif
CODPP	(r)	1	N	- déclaration de propriétés spécifiques des éléments du sous-ensemble CODENS (a)
LIBELLE	(s)	9	N	- ce code doit être discriminant
/	(8)	/	/	- ce chemin autorise la déclaration de plusieurs propriétés
CODPP	(t)	1	R	- code d'une propriété clé spécifique du sous-ensemble - ce code doit correspondre à celui d'une propriété déclarée en (r)
/	(9)	/	/	- ce chemin permet l'attribution de plusieurs clés à un sous-ensemble

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
/	(10)	/	/	- la description de clés spécifiques à un sous-ensemble est facultative
/	(11)	/	/	- la déclaration des propriétés spécifiques et l'attribution de clés à un sous-ensemble sont facultatives
CODPP	(u)	1	R	- code d'une propriété déclarée en (r) ou déclarée pour l'ensemble CODENS (d)° - cette propriété appartient à la sémantique de déclaration des éléments du sous-ensemble CODENS (a)
/	(12)	/	/	- le chemin permet la description successive de chaque propriété de cette sémantique de déclaration - cette sémantique de déclaration doit contenir successivement : - les codes des propriétés de la sémantique de déclaration des éléments de l'ensemble CODENS (d) - les codes des propriétés supports de la partition décrites en (e) - et éventuellement des codes de propriétés spécifiques décrites en (r)

Utilisations : . création d'un sous-ensemble d'un ensemble de base

```
cresou INFO * ensemble des étudiants du département informatique *
      dans ETUDIANT ayant DEPT égal 6 liste L50 * la valeur INFO * val*INFO** ras *
      taille 200
      dclpar NUMETUD
      dclpar NOM
      dclpar PRN
      dclpar DEPT
fcresou
```

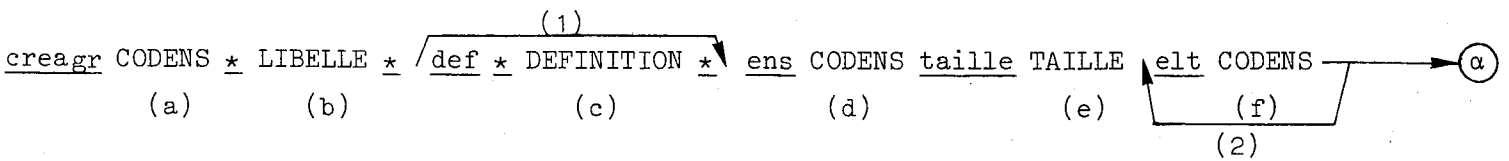
. création d'un sous-ensemble de relation

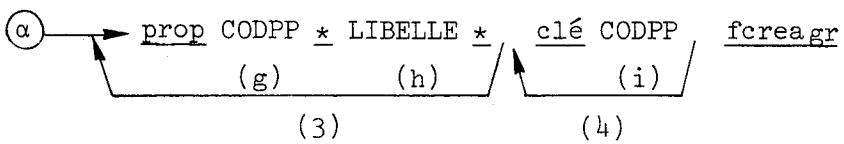
```
cresou INSCRSOIR * ensemble des inscriptions dans des UV en cours du soir *
      dans INSCRPT1 ayant NATURE égal 7 liste L51 * la valeur 1 * yal*1** 1 = cours du soir *
      dclpar CODINS1 dclpar NUMETUD dclpar CODUV
      dclpar NATURE
fcresou
```

III-2-1-5 - Instruction CREAGR

action : créer un agrégat et déclarer des propriétés de ses éléments

syntaxe :





validité :

Zone	Repere	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
CODENS	(a)	1	N	- code et libellé de l'agrégat à créer
LIBELLE	(b)	9	N	- ce code doit être discriminant
DEFINITION	(c)	10	N	- définition facultative (1) de l'agrégat CODENS (a). Une définition en extension est assurée en (f)
CODENS	(d)	1	R	- code de l'ensemble de référence de l'agrégat - c'est le code d'un ensemble de base ou d'une relation déjà créée
TAILLE	(e)	3	N	- nombre effectif d'éléments de l'agrégat
CODENS	(f)	1	R	- ce code est : - soit identique à CODENS (d) (si l'ensemble de référence est élément de l'agrégat) - soit le code d'un sous-ensemble déjà créé de l'ensemble CODENS (d)
/	(2)	/	/	- TAILLE (e) indique le nombre de fois où le chemin "elt CODENS" sera parcouru
CODPP	(g)	1	N	- déclaration d'une propriété des éléments de l'agrégat
LIBELLE	(h)	9	N	- ce code doit être discriminant
CODPP	(i)	1	R	- code d'une propriété déclarée en (g) qui est clé des éléments de l'agrégat
/	(3), (4)	/	/	- ces chemins autorisent des itérations sur la déclaration de propriétés et sur la définition de clés.

Remarquons que nous ne définissons pas de sémantique de déclaration des éléments d'un agrégat (cf. ch.II) ; aussi faudra-t-il prévoir une génération automatique des réalisations d'éléments d'un agrégat afin d'autoriser l'exécution de transactions de description.

Utilisations :

creagr PROMO * promotion * def * l'ensemble des étudiants de l'IUT constitue le seul élément de "PROMO" *
    ens ETUDIANT
    taille 1 elt ETUDIANT
    prop NB * nombre effectif d'étudiants *
    prop AGM * âge moyen d'un étudiant *
    prop CDPROMO * code de la promotion *
    prop ABS * durée moyenne de l'absence d'un étudiant *
    clé CDPROMO
fcreagr

creagr INFORMATICIENS * ras * def * ensemble d'un seul élément : l'ensemble des étudiants du département informatique *
    ens ETUDIANT
    taille 1 elt INFO
    prop NBD1 * nombre d'étudiants du département informatique *
    prop CDINF * code du département *
    prop PRIMD * pourcentage d'étudiants dont les parents travaillent dans le secteur primaire *
    clé CDINF
fcreagr

```
creagr DEPARTEMENT * ensemble des départements *
      ens ETUDIANT
      taille 5 elt INFO elt GEA elt STQ elt CJJ elt TCO elt SOC
      prop CDEPT * code du département *
      prop NBD2 * nombre d'étudiants d'un département *
      prop ABSD2 * durée moyenne d'absence d'un étudiant dans un département *
      prop AGMD2 * âge moyen d'un étudiant *
      clé CDEPT
fcreagr
```

- messages d'erreurs (numéros et libellés) :

```
280 * IL FAUT CITER LE CODE D'1 ENSEMBLE-BASE OU RELATION QUI EXISTE *
281 * LE NOMBRE DE DESCRIPTION D'ELT=TAILLE DE AGREGAT *
282 * IL FAUT CITER LE CODE D'1 ENSEMBLE QUI EXISTE *
283 * CE DOIT ETRE LE CODE D'1 ENSEMBLE-BASE OU RELATION OU SOUS-ENSEMBLE *
284 * SI CODE D'1 ENSEMBLE-BASE OU RELATION ALORS=ENS DE REFERENCE *
285 * UN ELEMENT D'AGREGAT SOUS-ENSEMBLE DOIT ETRE FILS DE L'ENSEMBLE DE REFERENCE *
```

III-2-2 - Instructions de modification et de suppression : MODENS, MODPP, SUPENS

Actuellement nous n'avons prévu que 3 instructions pour "modifier" les spécifications d'une structure logique de données :

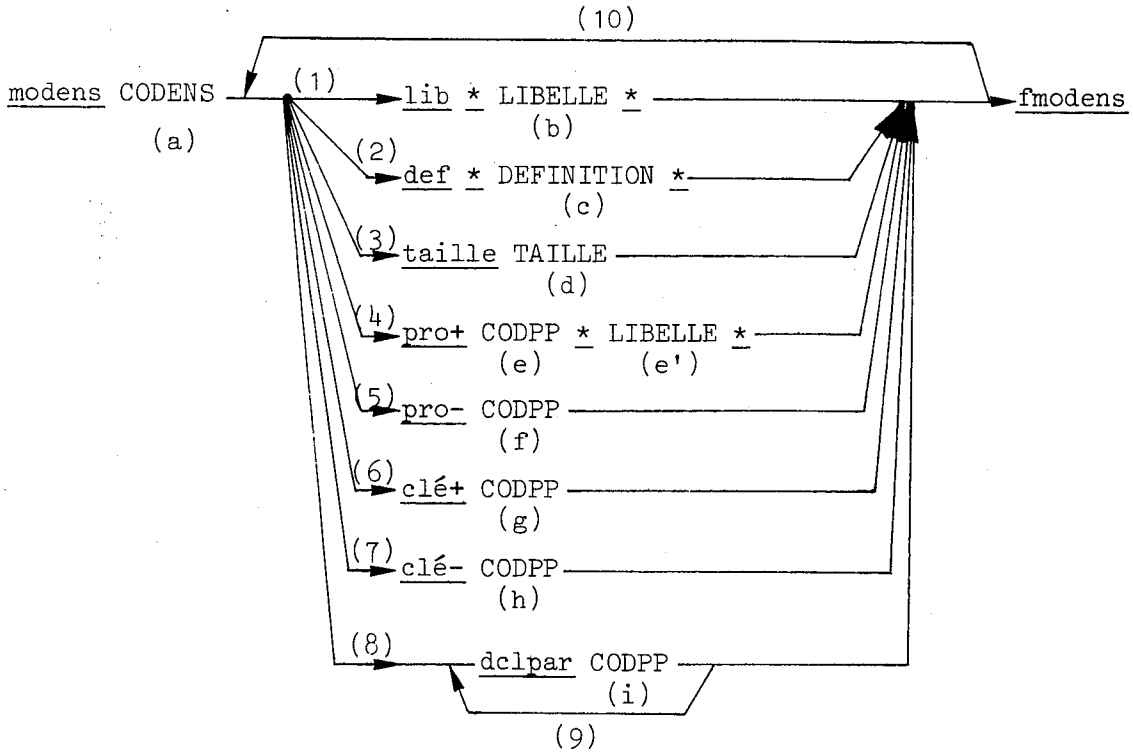
- MODENS pour modifier les caractéristiques d'un ensemble
- MODPP pour modifier les caractéristiques d'une propriété
- SUPENS pour supprimer les spécifications relatives à un ensemble.

L'utilisation d'une seule de ces 3 instructions ne permet pas de remettre en cause le schéma d'une structure logique de données, afin de toujours assurer la cohérence du modèle en fin d'exécution d'une instruction ; et ce sont souvent des utilisations combinées d'instructions de modification, de suppression et (ou) création qui autoriseront des changements plus fondamentaux dans une structure logique des données.

III-2-2-1 - Instruction MODENS

action : modifier une ou plusieurs caractéristiques d'un ensemble créé (ensemble de base, relation, sous-ensemble, ou agrégat)

syntaxe :



validité :

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
CODENS	(a)	1	R	- code pour identifier un ensemble déjà <u>créé</u> dans la structure logique des données - c'est le code d'un ensemble de base, d'une relation, d'un sous-ensemble ou d'un agrégat
/	(1),(2)	/	/	- ces chemins autorisent la spécification d'un nouveau LIBELLE ou d'une nouvelle DEFINITION quel que soit le type de l'ensemble concerné.
LIBELLE	(b)	9	N	- cette ligne remplace le libellé initial } les valeurs <u>u</u> et <u>ras</u> sont autorisées
DEFINITION	(c)	10	N	- ce texte remplace la définition initiale }
/	(3)	/	/	- ce chemin peut être emprunté si CODENS (a) est le code d'un ensemble de base, d'une relation, ou d'un sous-ensemble. Dans le cas d'un agrégat, la taille est définie par la description explicite de chacun de ses éléments (définition en extension d'un agrégat) et ne peut être modifiée
TAILLE	(d)	3	N	- nombre maximum d'éléments de l'ensemble - si CODENS (a) est le code d'un sous-ensemble, la TAILLE doit être inférieure ou égale à la taille de l'ensemble englobant - TAILLE doit être supérieure ou égale aux tailles éventuelles de sous-ensembles de l'ensemble CODENS (a) et doit être compatible avec les cardinaux minima et maxima de fonctions de dépendance associées à l'ensemble CODENS (a) argument de relations.
/	(4)	/	/	- chemin utilisable, quel que soit le type de l'ensemble à modifier, pour <u>déclarer</u> une nouvelle propriété
CODPP	(e)	1	N	- CODE et LIBELLE nécessaires à la déclaration d'une nouvelle propriété d'un ensemble
LIBELLE	(e')	9	N	- CODE doit être discriminant - dans le cas d'un sous-ensemble, c'est une propriété particulière à celui-ci.
/	(5)	/	/	- chemin de suppression d'une propriété d'un ensemble
CODPP	(f)	1	R	- ce doit être le code d'une propriété déclarée de l'ensemble CODENS (a) - la suppression est impossible si CODPP est une propriété : - clé unique de l'ensemble ou - appartenant à une sémantique de déclaration ou - support de partition dans la description d'un sous-ensemble.

				- si c'est une propriété composée, on supprime uniquement la propriété et la composition, et les composantes restent alors des propriétés simples de l'ensemble
/	(6)	/	/	- attribution d'une clé supplémentaire à un ensemble
CODPP	(g)	1	R	- code d'une propriété déclarée de l'ensemble
				- si la propriété a été décrite, elle doit être de type = 1 ou 5
/	(7)	/	/	- pour spécifier qu'une propriété n'est plus "clé" de l'ensemble
CODPP	(h)	1	R	- code d'une propriété clé de l'ensemble
				- cette suppression est impossible si : - c'est la clé unique de l'ensemble ou - si elle figure comme premier constituant de la sémantique de déclaration
				- la propriété n'est pas supprimée, elle n'est simplement plus clé de l'ensemble
/	(8)	/	/	- définition d'une nouvelle sémantique de déclaration d'un ensemble devant remplacer la précédente. Ce chemin ne peut pas être emprunté dans le cas d'un agrégat
CODPP	(i)	1	R	- code d'une propriété déclarée de l'ensemble
/	(9)	/	/	- description des divers constituants de la sémantique de déclaration
				- le premier constituant doit être une clé de l'ensemble
	(8)			- dans le cas d'une relation, ce chemin doit respecter les conditions indiqués p. III-18 (m) dans le cas d'un sous ensemble, voir p. III-23 (u)
				- si c'est un ensemble qui a des sous-ensembles, les sémantiques de déclaration de ceux-ci sont automatiquement modifiées en conséquence
/	(10)	/	/	- autorise plusieurs modifications avec la même instruction "modens"

Remarques :

- cette instruction peut être utilisée pour donner des valeurs à des caractéristiques indéfinies ('u') après création : (2) et (3)
- on ne modifie pas une sémantique de déclaration, mais on en donne une nouvelle (8)
- on ne supprime pas une propriété ou une clé appartenant à une sémantique de déclaration
- souvent, avant de supprimer, il faut déclarer les "objets" de remplacement
- on ne modifie pas les arguments d'une relation, ni les descriptions relatives aux propriétés supports définissant des sous-ensembles
- dans le cas de l'utilisation des chemins (5) ou (8), MACSI-P signale à l'utilisateur toutes les transactions qui deviennent invalides car elles utilisent ces propriétés supprimées ou cette sémantique de déclaration modifiée.

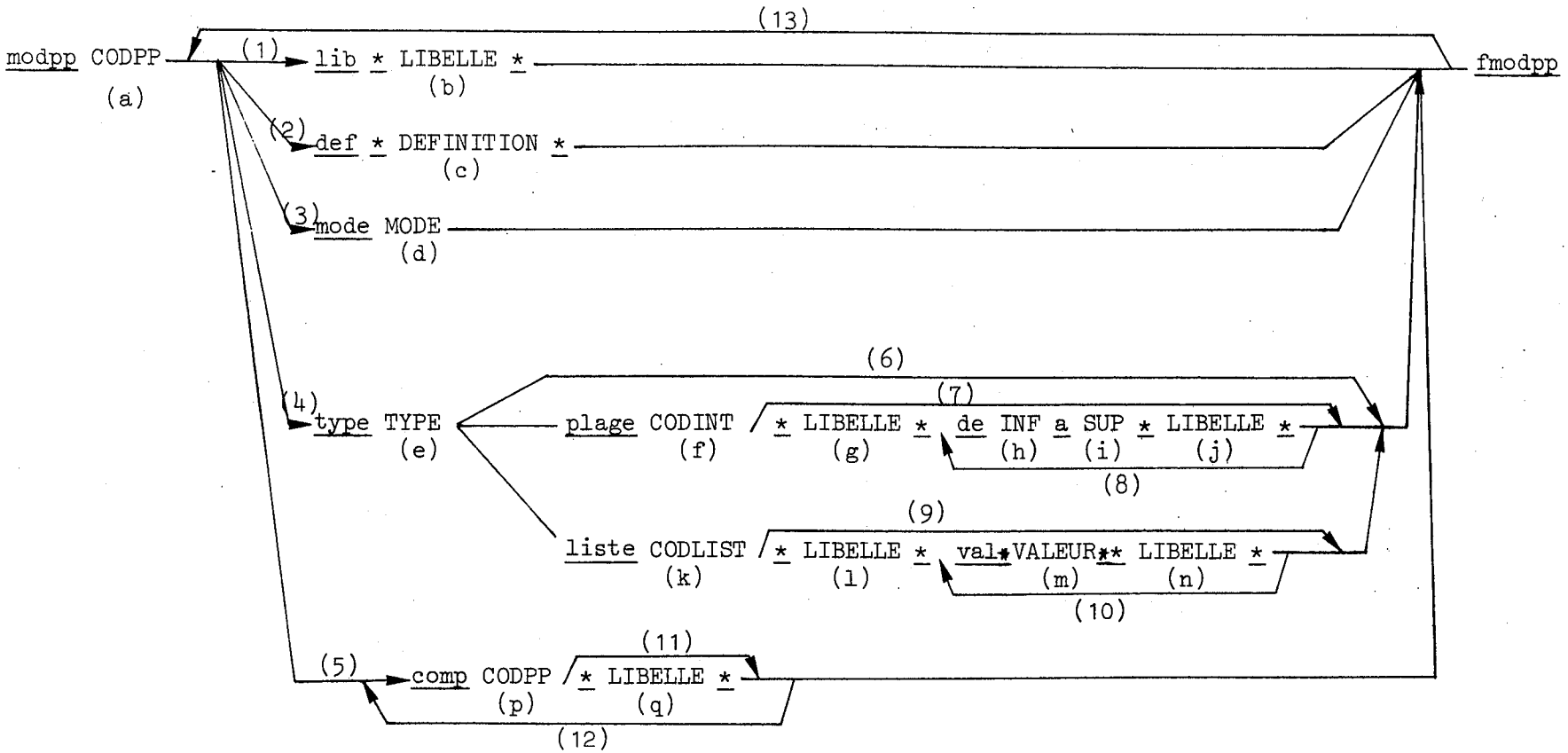
Utilisations : on veut donner une définition à l'ensemble FILIERE, modifier la taille et modifier la sémantique de déclaration en lui rajoutant la propriété LIBFIL.


```
modens FILIERE def * une filière est un cycle complet d'enseignement à l'issue duquel un diplôme national est délivré *
               taille 25
               dclpar CODFIL dclpar LIBFIL     } pour redécrire une nouvelle sémantique de déclaration de l'ensemble FILIERE
fmodens
```

III-2-2-2 - Instruction MODPP

action : modifier une ou plusieurs caractéristiques d'une propriété déclarée et décrite

syntaxe :



validité :

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
CODPP	(a)	1	R	- code d'une propriété déclarée et décrite dans la structure logique des données
LIBELLE	(b)	9	N	- donner un nouveau LIBELLE ou une nouvelle DEFINITION à la propriété
DEFINITION	(c)	10	N	
MODE	(d)	7	N	- attribuer un nouveau mode d'élaboration de la valeur de la propriété = 1 : saisie = 2 : calculée
/	(4)	/	/	- le chemin de modification du type d'une propriété est interdit si celle-ci est support de partitions
TYPE	(e)	7	N	- TYPE de la propriété = 1, 2, 3, 4, 5, 6, 7, 8, 9 ou 10 - si cette propriété est "clé", seuls les types 1 et 5 sont autorisés
/	(6)	/	/	- ce chemin est emprunté si TYPE (e) = 1, 2, 9 ou 10
plage	/	0	/	
CODINT	(f)	1	R ou N	- chemin pris si TYPE (e) = 3 ou 4
LIBELLE	(g)	9	N	- voir explications p. III-13
INF	(h)	3 ou 4	N	
SUP	(i)	3 ou 4	N	
LIBELLE	(j)	9	N	
liste	/	0	/	
CODLIST	(k)	1	R ou N	- chemin pris si TYPE (e) = 5, 6, 7 ou 8
LIBELLE	(l)	9	N	- voir explications pp. III-13, III-14
VALEUR	(m)	1,2,3 ou 4	N	
LIBELLE	(n)	9	N	
/	(5)	/	/	- ce chemin est utilisé pour dire que la propriété CODPP (a) est composée et décrire sa composition ; si cette propriété était déjà composée, la nouvelle composition remplace l'ancienne, mais les anciennes composantes restent propriétés simples de l'ensemble

Zone	Repere	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
CODPP	(p)	1	R ou N	- code et libellé d'une propriété composante de CODPP (a)
LIBELLE	(q)	9	N	- si la propriété est déjà déclarée, alors il faut emprunter le chemin (11), et dans ce cas, CODPP (p) est le code d'une propriété appartenant au même ensemble que la propriété CODPP (a)
				- si c'est la déclaration d'une nouvelle propriété : CODPP doit être discriminant, et il faut indiquer le libellé
/	(12)	/	/	- ce chemin permet la description de toutes les composantes d'une propriété composée
/	(13)	/	/	- pour faire plusieurs modifications sur une propriété lors d'une même instruction.

Remarques :

- On ne peut pas modifier l'appartenance d'une propriété à un ensemble avec l'instruction "modpp", ceci sera fait par des utilisations de l'instruction "modens".
- Dans le cas où les chemins 3, 4 ou 5 sont employés, l'exécution de l'instruction se termine par l'édition de la liste des transactions qui ne sont plus valides, car l'utilisation et les contrôles de la propriété ne sont peut-être plus compatibles avec les descriptions déjà faites de transactions.

Utilisations :

Donner une définition à la propriété DEPT décrite p. III-16 ; il est interdit par exemple de modifier sa liste de **valeurs**, car c'est une propriété support de partition de l'ensemble ETUDIANT.

```
modpp DEPT def * l'UER IUT2 est organisée en départements jouissant d'une partielle autonomie administrative
                 et pédagogique *
fmodpp
```

Modifier l'intervalle de définition de la CAPACITE d'une salle (p. III-16)

```
modpp CAPACITE type 3 plage I631 * capacité d'une salle *
                      de 1 à 1000 * ras *
fmodpp
```

III-2-2-3 - Instruction SUPENS

action : supprimer un ensemble créé (ensemble de base, sous-ensemble, relation ou agrégat).

syntaxe : supens CODENS fsupens

validité :

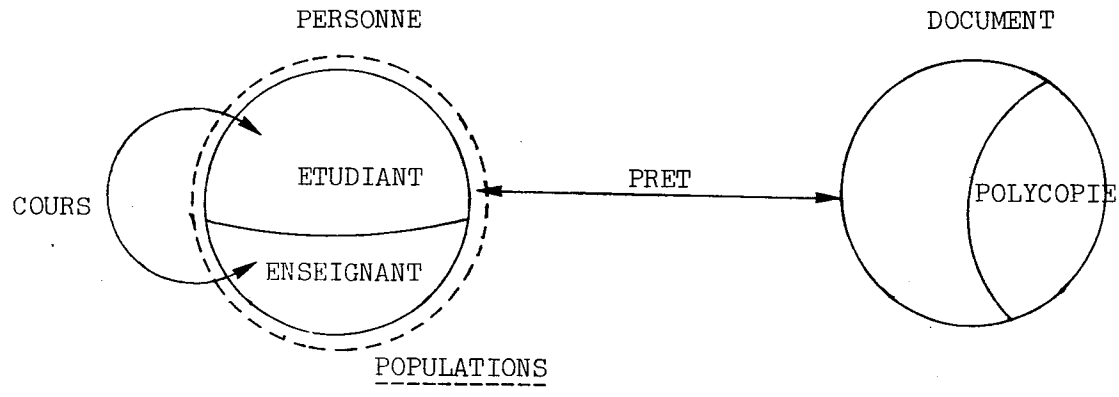
- CODENS doit être le code (valeur de type 1) d'un ensemble de base, d'une relation, d'un sous-ensemble ou d'un agrégat créé
- cet ensemble ne doit être argument d'aucune relation, ni élément, ni ensemble de référence d'aucun agrégat et n'avoir aucun sous-ensemble spécifié.

effets :

- cette instruction supprime toutes les spécifications relatives à l'ensemble (code, libellé, définition, taille, clés, ...) et toutes ses propriétés et leurs spécifications.
- MACSI-P indique toutes les transactions devenues ainsi invalides.

utilisations :

- Soient des ensembles créés :
  - . les ensembles de base PERSONNE et DOCUMENT
  - . les sous-ensembles ETUDIANT et ENSEIGNANT de PERSONNE et POLYCOPIE de DOCUMENT
  - . les relations PRET (PERSONNE, DOCUMENT) et COURS (ETUDIANT, ENSEIGNANT)
  - . l'agrégat POPULATIONS/PERSONNE {ETUDIANT, ENSEIGNANT}



- On peut alors écrire les instructions suivantes dans un ordre quelconque :

  supens POPULATIONS fsupens
  et/ou supens PRET fsupens
  et/ou supens COURS fsupens
  et/ou supens POLYCOPIE fsupens

- Si l'on veut supprimer l'ensemble DOCUMENT, on doit écrire la liste ordonnée d'instructions :

  supens POLYCOPIE fsupens
  et supens PRET fsupens
  et supens DOCUMENT fsupens

- Si l'on veut supprimer l'ensemble ENSEIGNANT, on doit écrire les instructions :

  supens POPULATIONS fsupens
  et supens COURS fsupens
  et supens ENSEIGNANT fsupens

- Si l'on veut supprimer l'ensemble PERSONNE, on doit écrire les instructions :

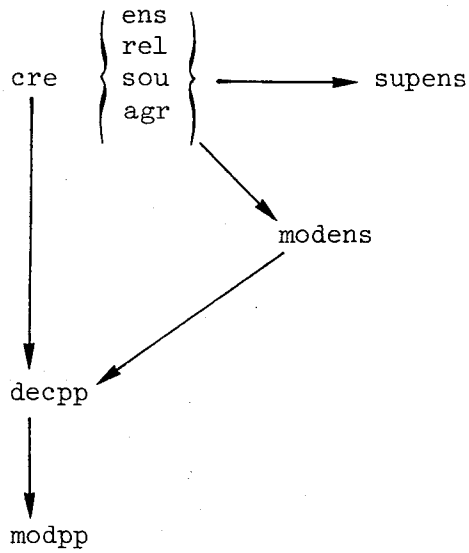
  supens POPULATIONS fsupens
  et supens COURS fsupens
  et supens ENSEIGNANT fsupens
  et supens ETUDIANT fsupens
  et supens PRET fsupens
  et supens PERSONNE fsupens

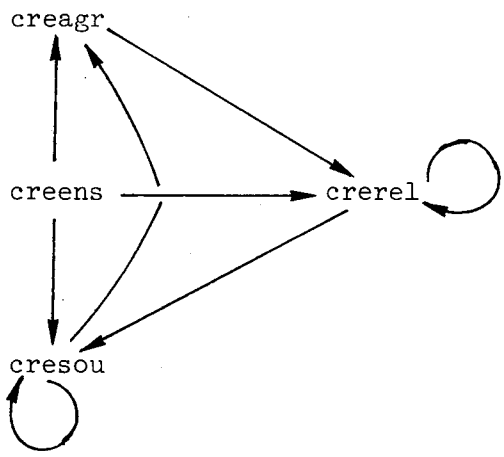
Remarque :

Cette solution garantit qu'un utilisateur ne peut supprimer un ensemble que s'il connaît toutes les conséquences de cette suppression.

III-2-3 - Conclusion

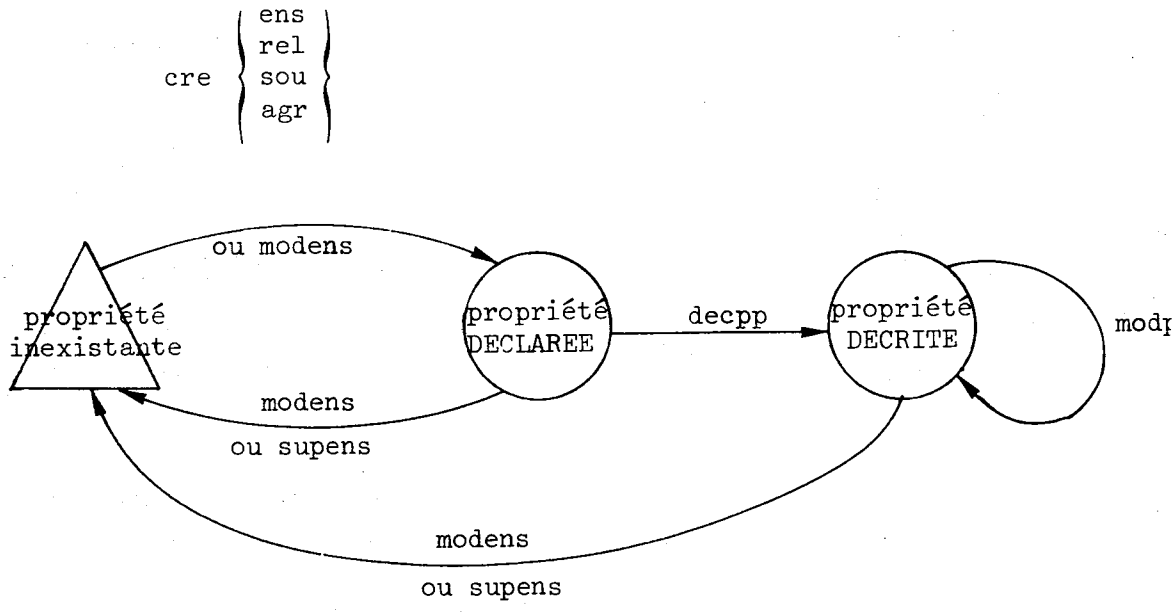
Le lecteur a pu constater que les différentes instructions ne peuvent pas s'utiliser dans n'importe quel ordre : on n'a pas le droit, par exemple, de spécifier des propriétés sans avoir décrit au préalable les ensembles d'éléments caractérisés par ces propriétés. Le processus considéré comme cohérent pour spécifier une structure logique de données par MACSI-P peut être résumé dans les figures suivantes :





Commentaires :

- Les flèches continues indiquent l'ordre dans lequel les instructions peuvent être utilisées. L'utilisation des instructions situées à l'origine de flèches entraînent la déclaration ou création de composants qui seront pris en compte par les instructions pointées.
  Par exemple, la description d'une propriété (decpp) ne peut être effectuée que si cette propriété a été déclarée par une création ou une modification (pro+) d'un ensemble. On ne peut créer une relation (crerel) que si ses ensembles-arguments ont été créés au préalable en tant qu'ensemble de base, sous-ensembles, relations et/ou agrégats.
  C'est dans une optique de conception modulaire et descendante que ces contraintes ont été adoptées.
- Ces figures sont à combiner avec les graphes d'états donnés p. III-4 ; donnons cette combinaison pour une propriété :



III-3 - INSTRUCTIONS DE SPECIFICATION DE TRANSACTIONS

III-3-1 - Caractéristiques générales des transactions

Une approche qualitative de la définition d'une transaction est faite dans le § II-2-3. Nous pouvons rappeler brièvement le contenu des spécifications dans le cas d'une transaction d'entrée.

Spécifications fonctionnelles d'une transaction d'entrée :

Nous les avons séparées en 3 parties :

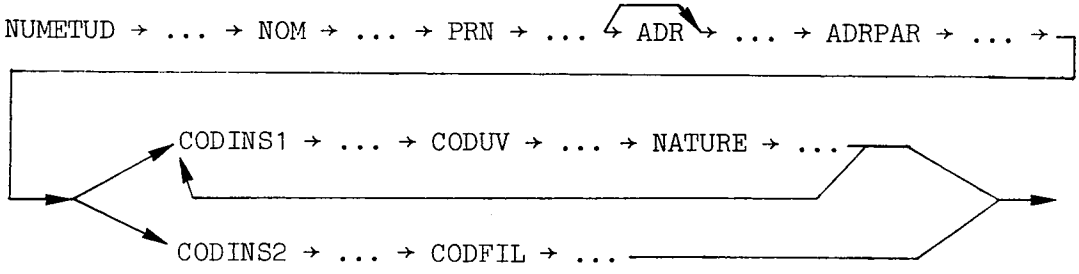
- La *première partie* décrit le CODE discriminant de cette transaction, les divers commentaires documentaires (LIBELLE et DEFINITION), ainsi que la désignation de son ENSEMBLE de REFERENCE et de l'opération envisagée (déclaration, citation, création, description, modification ou suppression).

- La *deuxième partie* est relative au PROLOGUE de la transaction qui commence éventuellement par la spécification d'un programme interne, et qui contient obligatoirement un chemin d'identification ou un chemin de déclaration de l'OBJET de la transaction.

- La *troisième partie* est décrite par le schéma de la transaction.

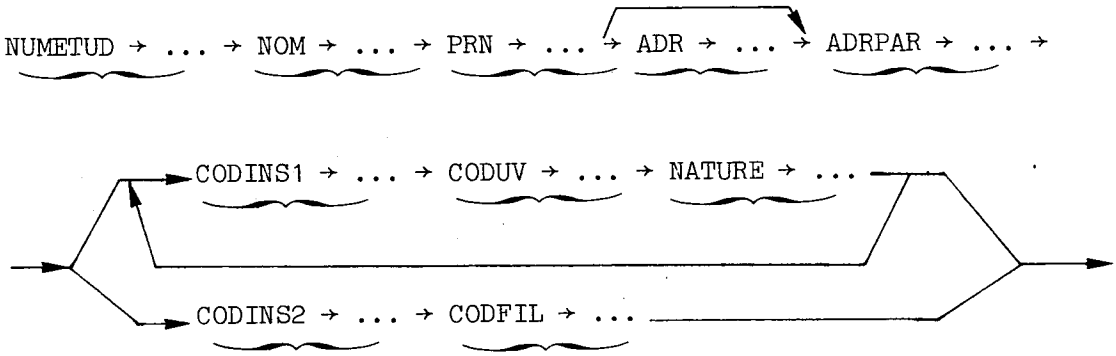
Description du graphe d'un schéma de transaction : un graphe des champs

Prenons l'exemple suivant pour présenter une solution pour décrire un schéma de transaction.

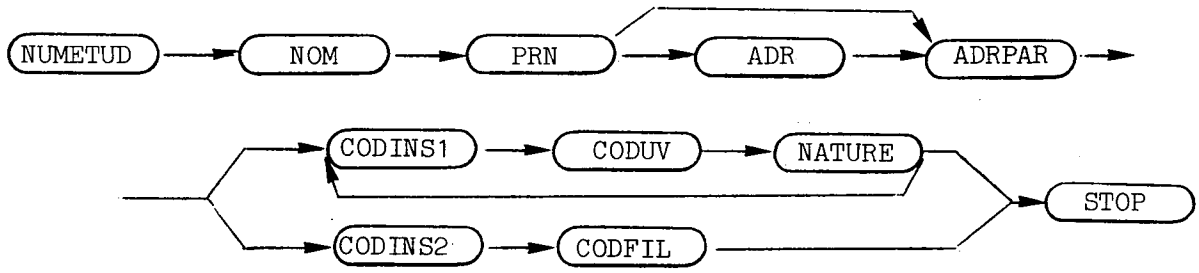


Ce schéma de transaction est un graphe orienté dont la construction est soumise à des règles strictes, cf.§II-2-3, et dont l'unité de description est le CHAMP (→ code d'une propriété → ... →).

Une représentation interne de ce schéma de transaction suppose avant tout une description du "graphe des champs" où un noeud du graphe est désigné par le code de la propriété décrivant le champ (code du champ).



Nous obtenons alors le "graphe des champs" :



Nous pouvons remarquer que nous n'avons utilisé que 3 figures élémentaires pour construire un tel graphe.

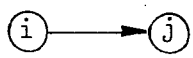
Figure 1 : 

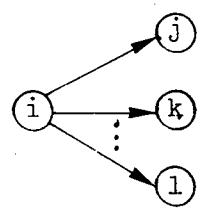
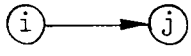
Figure 2 : 

Figure 3 : 

Un langage de description d'un graphe :

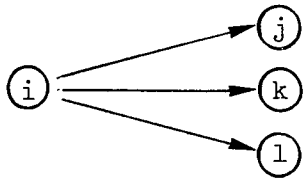
Aussi, pour rédiger les spécifications relatives à un schéma de transaction, avons-nous introduit "un langage externe" de description du graphe, basé sur 3 règles qui correspondent aux 3 figures ci-dessus.

. Succession de champs :



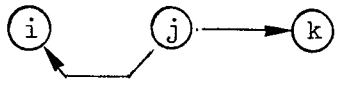
⟹ règle 1 : "après champ i faire champ j"

. Choix entre champs :



==> règle 2 : "après champ i faire champ j
ou champ k
ou champ l"

. Itération sur un champ :



==> règle 3 : "après champ j faire champ k
ou refaire champ i"

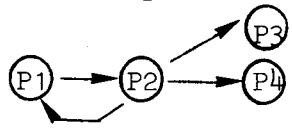
. Stop est le champ final ; il ne correspond à aucune lecture de données.

Ainsi, avec ce langage, le graphe précédent s'écrit :

```
après NUMETUD faire NOM
après NOM     faire PRN
après PRN     faire ADR ou ADRPAR
après ADR     faire ADRPAR
après ADRPAR  faire CODINS1 ou CODINS2
après CODINS1 faire CODUV
après CODUV   faire NATURE
après NATURE  faire stop ou refaire CODINS1
après CODINS2 faire CODFIL
après CODFIL  faire stop
```

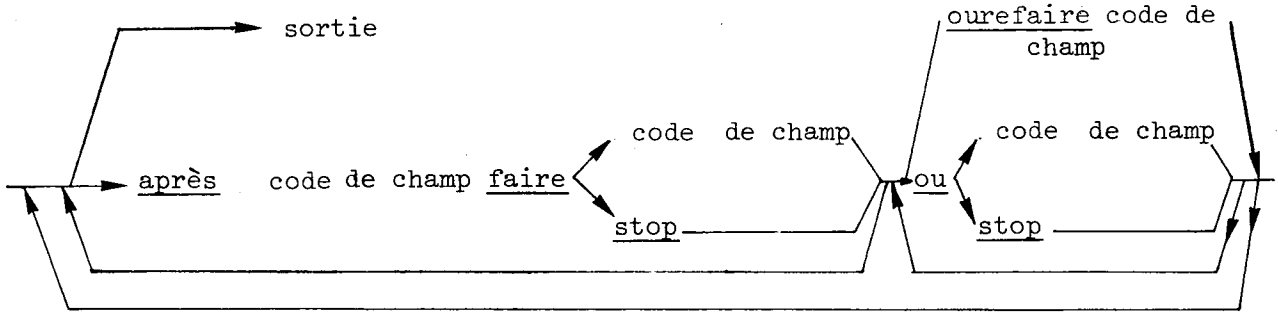
Nous pouvons remarquer que :

- Il y a autant de faire, ou et ourefaire que d'arcs dans le graphe.
- La règle 3 permet d'affirmer qu'un circuit du graphe admet toujours au moins un champ pour sortir de l'itération. Cette écriture permet de décrire une itération sur un champ de début d'un chemin lié itératif.

Le graphe des champs : 

s'écrit : après P1 faire P2
après P2 faire P3
ou P4
ourefaire P1

- L'ordre d'écriture des règles "après ..." n'est pas indifférent : la règle 2 ne peut pas être remplacée par plusieurs utilisations de la règle 1.
- Un même graphe ne peut pas avoir deux noeuds qui ont le même code de champ.
- La grammaire de ce langage peut être représentée par l'automate suivant :



Descriptions fonctionnelles d'un schéma de transaction : un graphe renseigné

Nous venons de voir que la "structure" d'un schéma de transaction est entièrement définie par son graphe de champs. Pour assurer l'ensemble des spécifications de tels schémas, il suffit de définir divers "renseignements" sur chaque champ et ainsi obtenir un "graphe renseigné" équivalent fonctionnellement à la transaction concernée.

Renseignements à fournir sur le graphe des champs

- Pour chaque champ est indiqué le composant touché dans la structure logique de données, désigné par le CODE de la PROPRIETE qui identifie ainsi :
  - . la valeur à lire
  - . le contrôle standard à effectuer sur cette valeur lue.

- Pour un champ simple (cf. § II-2-3) décrit par une propriété non encore déclarée de l'ensemble de référence de la transaction, il faut indiquer son LIBELLE.

- Pour chaque champ, il est possible de décrire un ou plusieurs CONTROLES PARTICULIERS (cf. § II-4-2-1) dont on donne pour chacun :
  - . le CODE et la DEFINITION du programme de contrôle
  - . le CODE et le DIAGNOSTIC de l'erreur associée.

D'une façon générale, les renseignements à fournir pour un graphe des champs sont :

- des *renseignements structuraux obligatoires*
  - . CODE de la propriété
  - . LIBELLE d'une nouvelle propriété à introduire dans la structure logique de données
- ou des *renseignements facultatifs* sur les CONTROLES PARTICULIERS de certains champs.

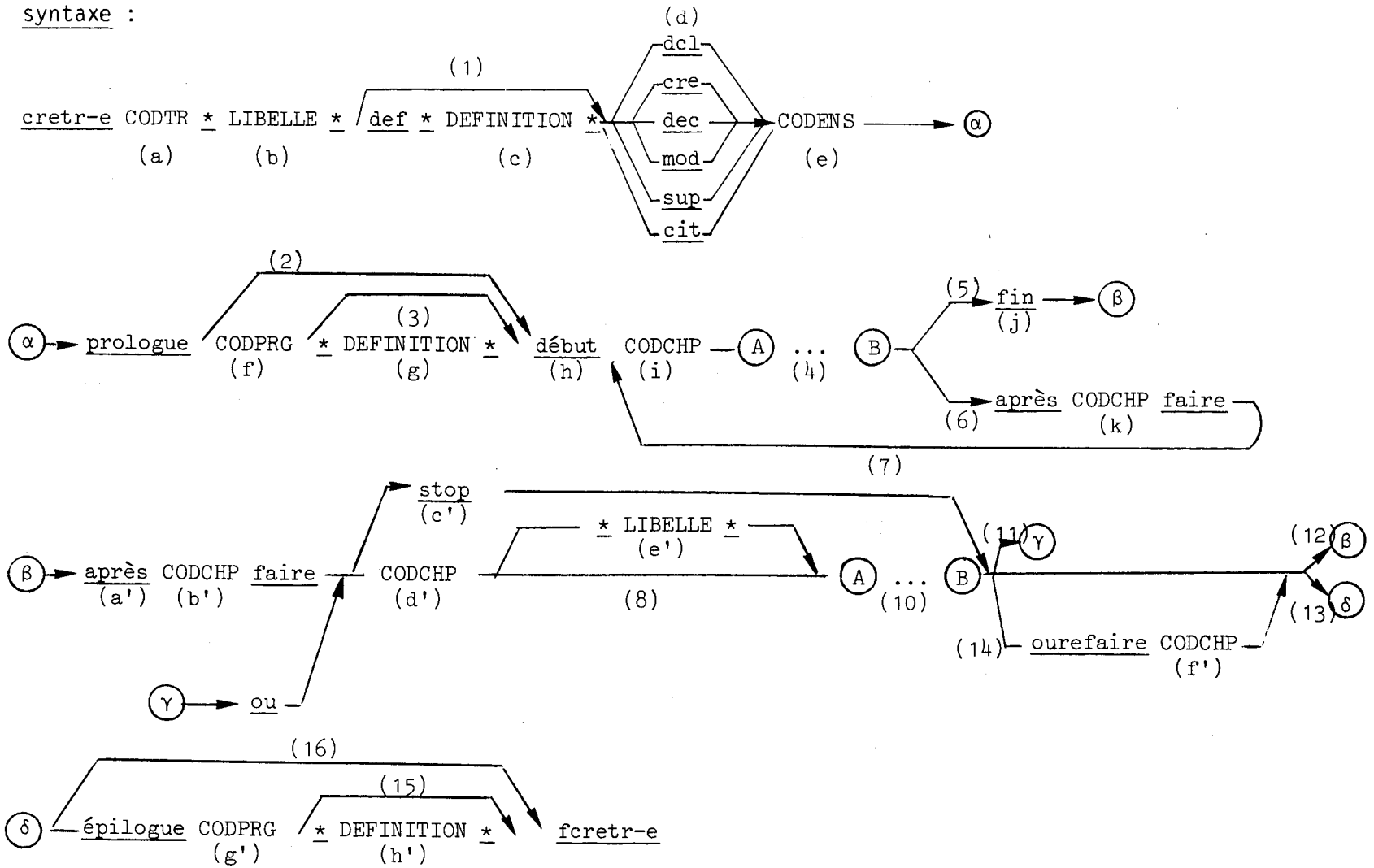
Exemples : Renseignements du graphe précédent.

RENSEIGNEMENTS STRUCTURAUX		CONTROLES PARTICULIERS
CODE de CHAMP	Commentaires	
NUMETUD	/	vérifier que les deux premiers caractères = l'année en cours
NOM	/	/
PRN	/	/
ADR	/	/
ADRPAR	*adresse des parents* (nlle propriété de ETUDIANT)	
CODINS1	. création d'un nouveau lien INSCRPT1 (ETUDIANT, UV)	champs non simples d'un chemin lié itératif
CODUV	. (appartient à la sémantique de déclaration d'un lien INSCRPT1)	
NATURE	. de INSCRPT1	
CONSINS2	. création d'un nouveau lien INSCRPT2 (ETUDIANT, FILIERE)	champs non simples d'un chemin lié
CODFIL	. (appartient à la sémantique de déclaration d'un lien INSCRPT2)	

Les instructions de création d'une transaction d'entrée ou de création d'une transaction de sortie assurent les spécifications du graphe des champs et des renseignements associés.

action : créer une transaction d'entrée et décrire (renseigner) tous les champs de son schéma.

syntaxe :



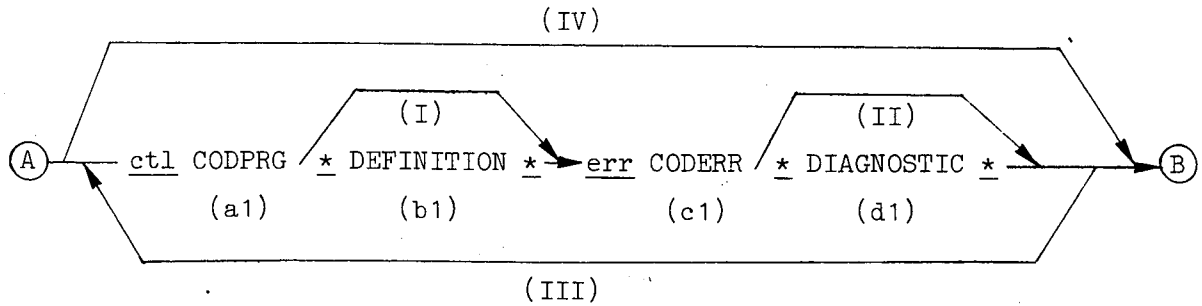
validité :

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
CODTR	(a)	1	N	- CODE, LIBELLE et DEFINITION d'une nouvelle transaction d'entrée à CREER
LIBELLE	(b)	9	N	- CODTR doit être discriminant
DEFINITION	(c)	10	N	
/	(1)	/	/	- la définition d'une transaction est facultative
dcl,cre,dec, mod,sup,cit	(d)	0	/	- choix d'une opération de base relative à un élément de l'ENSEMBLE de REFERENCE - dcl=déclaration, cré=création, dec=description, mod=modification, sup=suppression,cit=citation
CODENS	(e)	1	R	- code de l'ensemble de référence de la transaction - ce doit être le code d'un ensemble déjà créé dans la structure logique des données
prologue	/	0	/	- début du prologue de la transaction d'entrée ; ce prologue s'achève en (j) = fin
CODPRG	(f)	1	R ou N	- CODE et DEFINITION d'un programme interne dont l'exécution doit précéder celle de la transaction
DEFINITION	(g)	10	N	
/	(2)	/	/	- chemin utilisé si la transaction ne doit pas commencer par un traitement interne
/	(3)	/	/	- chemin emprunté si CODPRG (f) est le CODE d'un programme déjà déclaré, sinon la définition est obligatoire
début	(h)	0	/	- début et fin du chemin de déclaration si (d) = cré ou dcl ou du chemin d'identification si (d) = mod, dec, sup ou cit
fin	(i)	0	/	
CODCHP	(i)	1	R	- code d'un champ du chemin d'identification ou de déclaration - un chemin d'identification ou de déclaration est un chemin simple (sans boucles, ni choix). - ce doit être le code d'une propriété déjà déclarée dans la structure logique de données - cette propriété doit être une clé de l'ensemble CODENS (e) (dans le cas du premier passage sur ce chemin) ou appartenir à la sémantique de déclaration de cet ensemble (pour les autres itérations (7) sur ce chemin)
(A) ... (B)	(4)	/	/	- voir définition de ce chemin p. III-55
/	(5)	/	/	- fin du chemin d'identification ou de déclaration, et fin du prologue de la transaction
fin	(j)	0	/	- le chemin (5) est pris obligatoirement si (d) = dec, mod, sup ou cit

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
/	(6)	/	/	- ce chemin n'est pris que si (d) = cre ou dcl ; il permet de continuer la description du chemin de déclaration
CODCHP	(k)	1	R	- code d'un champ déjà décrit en CODCHP (i), et dont on veut définir le successeur
/	(7)	/	/	- pour décrire le champ CODCHP (i) successeur du champ CODCHP (k)
	(β)			
après	(a')	0	/	- début de la description d'un arc du champ CODCHP (b') (déjà décrit) vers le champ CODCHP (d') (à décrire)
CODCHP	(b')	1	R	- code d'un champ déjà décrit en CODCHP (d') et dont on va définir le successeur - dans le cas du premier passage sur ce chemin "(β)→...", c'est le code du dernier champ CODCHP (i) du prologue
stop	(c')	0	/	- pour indiquer que le champ CODCHP (b') est un champ final, un champ sans successeur
CODCHP	(d')	1	R ou N	- code d'un champ successeur du champ CODCHP (b') - ce doit être le code d'une propriété déclarée ou à déclarer dans la structure logique de données
/	(8)	/	/	- si CODCHP (d') est le code d'une propriété déjà déclarée
/	(9)	/	/	- si CODCHP (d') est le code d'une nouvelle propriété à déclarer pour l'ensemble CODENS (e)
LIBELLE	(e')	9	N	- libellé d'une nouvelle propriété de code CODCHP (d') de CODENS (e)
(A) ... (B)	(10)	/	/	- voir définition de ce chemin p. III-55
/	(11)	/	/	- ce chemin est pris si le champ CODCHP (b') a plusieurs successeurs (CODCHP (d') ou stop (c')) - ce chemin autorise la description d'un chemin multiple pour le schéma de la transaction
/	(12)	/	/	- chemin emprunté pour décrire un nouvel arc du graphe des champs
/	(13)	/	/	- chemin utilisé si le graphe des champs est totalement décrit et renseigné ; il est obligatoire que stop (c') fut successeur d'au moins un champ CODCHP (b')
/	(14)	/	/	- chemin utilisé pour construire une itération sur le champ CODCHP (f')

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
CODCHP	(f')	1	R	- code d'un champ déjà décrit en CODCHP (d'), marquant le début d'un chemin lié itératif
CODPRG	(g')	1	R ou N	- CODE et DEFINITION d'un programme interne dont l'exécution doit suivre celle de la transaction
DEFINITION	(h')	10	N	
/	(15)	/	/	- chemin utilisé si CODPRG (g') est le code d'un programme déjà déclaré, sinon la définition est obligatoire
/	(16)	/	/	- la description d'un épilogue d'une transaction est facultative.

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
(A) ... (B)	(4)ou(10)	/	/	- chemin de définition d'un ou plusieurs CONTROLES PARTICULIERS à associer au champ en cours de description - c'est le graphe :



CODPRG	(a1)	1	R ou N	- code d'un programme de contrôle créé ou à créer
DEFINITION	(b1)	10	N	- DEFINITION du programme CODPRG (a1) à créer
/	(I)	/	/	- si CODPRG (a1) est le code d'un programme déjà créé
CODERR	(c1)	1	R ou N	- code de l'erreur à associer au programme CODPRG (a1)
DIAGNOSTIC	(d1)	10	N	- DEFINITION du message d'une nouvelle erreur CODERR à créer
/	(II)	/	/	- cas d'une erreur déjà créée
/	(III)	/	/	- il est possible d'associer plusieurs contrôles à un même champ
/	(IV)	/	/	- un champ peut n'avoir aucun CONTROLE PARTICULIER associé.

Utilisation :

- spécifications de la transaction décrite § III-3-1.

```
cretr-e T1 * création et inscription d'un étudiant *
          def * un étudiant peut s'inscrire dans une seule filière ou dans plusieurs unités de valeurs *
      cre ETUDIANT
      prologue début                NUMETUD ctl P147 err E1470
               après NUMETUD faire NOM
               après NOM     faire PRN
               fin
      après PRN faire ADR
                  ou ADRPAR * adresse des parents de l'étudiant *
      après ADR faire ADRPAR
      après ADRPAR faire CODINS1 ctl P200 * clé d'un nouvel élément de la relation ... * err E200 * ... *
                    ou CODINS2 ctl P300 * ... * err E300 * ... *
      après CODINS1 faire CODUV  ctl P250 * composante de la clé CODINS1 * err E250 * ... *
      après CODUV   faire NATURE
      après NATURE faire stop
                  ourefaire CODINS1
      après CODINS2 faire CODFIL
      après CODFIL  faire stop
    épilogue P170 * éditer les informations qui viennent d'être enregistrées après exécution d'une réalisation de T1 *
fcretr-e
```

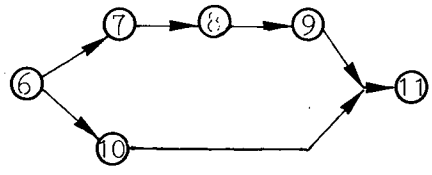
III-3-3 - Création d'une transaction de sortie

Remarques préalables :

Les descriptions qualitatives des spécifications fonctionnelles des transactions de sortie sont faites au § II.2.3 ; nous pouvons rappeler brièvement leurs particularités essentielles :

- Une transaction de sortie a pour OBJET :
  - . un élément de l'ensemble de référence ; alors un chemin d'identification (lecture de la clé) est obligatoire.
  - . tout élément de l'ensemble de référence ; dans ce cas, la deuxième partie des spécifications est vide, il n'y a pas d'identification nécessaire de l'objet.
- Un champ d'une transaction de sortie peut nécessiter un CALCUL interne avant l'édition.
- Le schéma de la transaction (décrit par un graphe des champs suivant les règles définies p. § III-3-1) peut contenir des chemins multiples : ceux-ci doivent être précédés de programmes internes d'AIGUILLAGE qui, lors d'une réalisation (exécution) de la transaction, sélectionnent un chemin unique.

Exemple : Soit le graphe des champs :



...

après 6 faire 7
ou 10
....

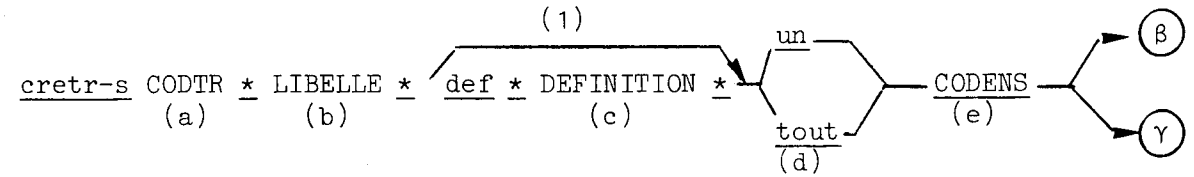
Le champ 6 doit être suivi d'un programme d'aiguillage sur 7 ou 10 :

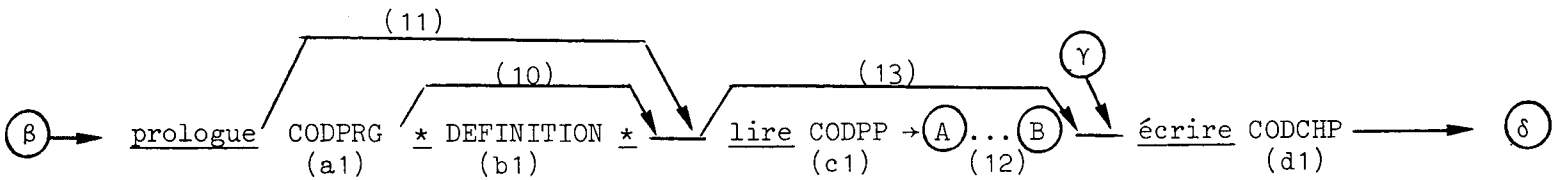
après 6 <programme d'aiguillage> faire 7
ou 10

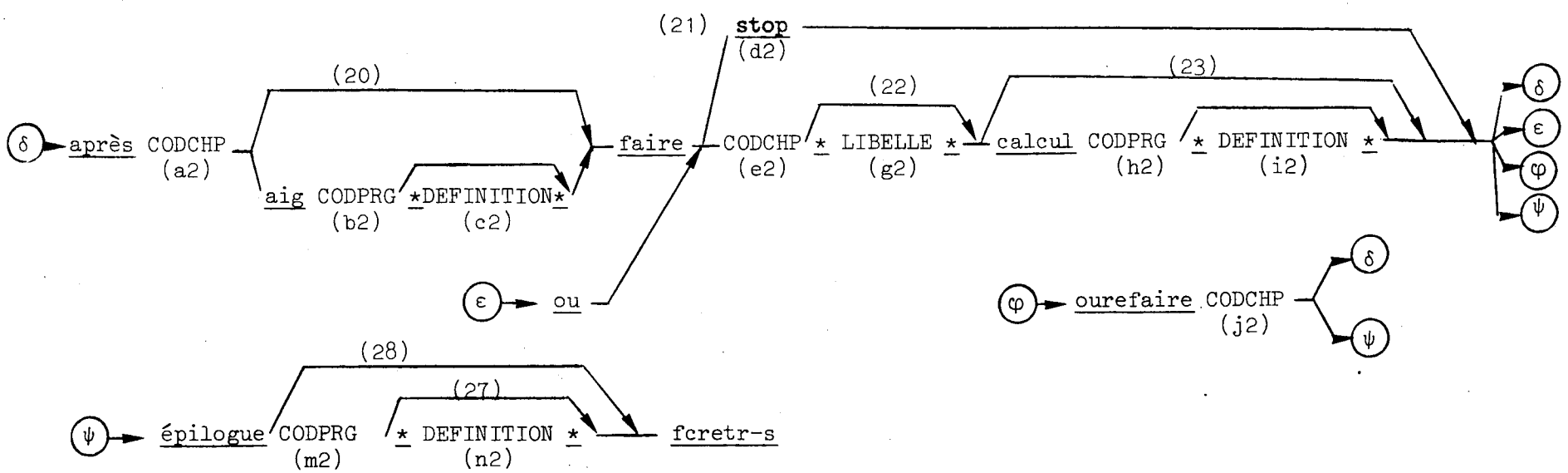
- Instruction CRETR-S

action : créer une transaction de sortie et décrire (renseigner) tous les champs de son schéma.

syntaxe :







validité :

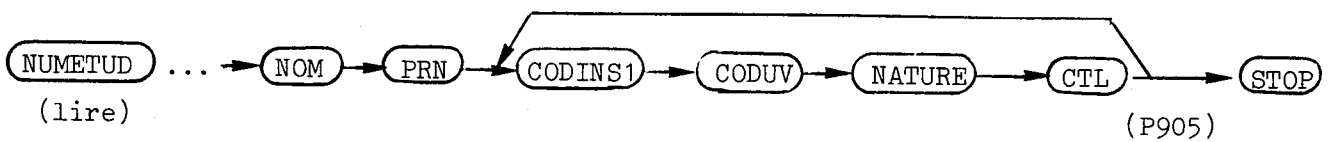
Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
CODTR	(a)	1	N	- code, libellé et définition d'une nouvelle transaction de sortie à CREER
LIBELLE	(b)	9	N	- CODTR doit être discriminant
DEFINITION	(c)	10	N	- la définition est facultative (1)
un	(d)	0	/	- la transaction de code CODTR (a) s'exécutera sur un seul élément de l'ensemble de référence
tout	(d)	0	/	- la transaction de code CODTR (a) s'exécutera sur tout élément de l'ensemble de référence
CODENS	(e)	1	R	- code de l'ensemble de référence de la transaction
/	(β)	/	/	- chemin pris si la zone (d) = un
	(γ)			- chemin utilisé si la transaction ne comporte aucun prologue
prologue	/	0	/	- début du prologue de la transaction
CODPRG	(a1)	1	R, N	- code et définition d'un programme interne dont l'exécution doit précéder celle de
DEFINITION	(b1)	10	N	la transaction
/	(10)	/	/	- chemin emprunté si CODPRG (a1) est le code d'un programme déjà déclaré
/	(11)	/	/	- si le prologue ne contient aucun programme interne
CODPP	(c1)	1	R	- code d'une propriété clé de l'ensemble CODENS (e), pour identifier l'objet de la transaction
(A)...(B)	(12)	/	/	- voir définition de ce chemin, p. III-55
/	(13)	/	/	- si zone (d) = tout : il n'y a pas de champ d'identification
/	(γ)	/	/	- début du graphe des champs d'édition
CODCHP	(d1)	1	R	- code du premier noeud du graphe des champs
				- code d'une propriété déjà déclarée de l'ensemble de référence
/	(δ)	/	/	- début de la description d'un arc du graphe des champs
CODCHP	(a2)	1	R	- numéro d'un champ déjà décrit en CODCHP (d1) ou en CODCHP (e2) et dont on veut définir le successeur
/	(20)	/	/	- si le champ CODCHP (a2) n'est pas suivi d'un chemin multiple

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
aig	/	0	/	- désignation ou déclaration d'un programme d'aiguillage sur chemin multiple après le champ CODCHP (a2)
CODPRG	(b2)	1	R, N	
DEFINITION	(c2)	10	N	
faire	/	0	/	- début de la description du successeur de CODCHP (a2)
/	(21)	/	/	- si CODCHP (a2) était un champ final du graphe (champ sans successeur)
stop	(d2)	0	/	
CODCHP	(e2)	1	R ou N	- code du champ successeur du champ CODCHP (a2) - ce doit être le code d'une propriété déclarée ou à déclarer
/	(22)	/	/	- si CODCHP (e2) est le code d'une propriété déjà déclarée
/	(23)	/	/	- si la valeur de la propriété CODCHP (e2) n'est pas à calculer pour l'élément objet de la transaction
LIBELLE	(g2)	9	N	- libellé d'une nouvelle propriété calculée à déclarer pour l'ensemble CODENS (e)
CODPRG	(h2)	1	R, N	- code et définition d'un programme interne de calcul de la valeur de la propriété CODCHP (e2) pour un élément - la définition ne doit pas être donnée si le programme a déjà été déclaré auparavant
DEFINITION	(i2)	10	N	
/	(δ)	/	/	- s'il reste encore des arcs du graphe à décrire
/	(ε)	/	/	- si le champ CODCHP (a2) a plusieurs successeurs
/	(ψ)	/	/	- si le graphe des champs est totalement décrit et renseigné
/	(φ)	/	/	- si le champ CODCHP (a2) marque la fin d'un chemin lié itératif : un de ses successeurs (j2) est le début de ce chemin
CODCHP	(j2)	1	R	- code d'un champ déjà décrit en CODCHP (e2), marquant le début d'un chemin lié itératif
CODPRG	(m2)	1	R, N	- code et définition d'un programme interne dont l'exécution doit suivre celle de la transaction
DEFINITION	(n2)	10	N	
/	(27)	/	/	- si CODPRG est le code d'un programme déjà déclaré
/	(28)	/	/	- l'épilogue est facultatif.

Utilisation : spécifications d'une transaction de sortie

```
cretr-s T3 * établir la liste des UV où s'est inscrit un étudiant *
       un ETUDIANT
       prologue lire NUMETUD ctl P900 * vérifier que cet étudiant s'est inscrit dans au moins une UV *
                             err E900 * cet étudiant n'est inscrit dans aucune UV *
       écrire NOM
       après NOM faire PRN
       après PRN faire CODINS1
       après CODINS1 faire CODUV
       après CODUV faire NATURE
       après NATURE faire CTL
       après CTL aig P905 * arrêt ("stop") si toutes les inscriptions d'un étudiant ont été épuisées *
                faire stop
                ourefaire CODINS1
fcretr-s
```

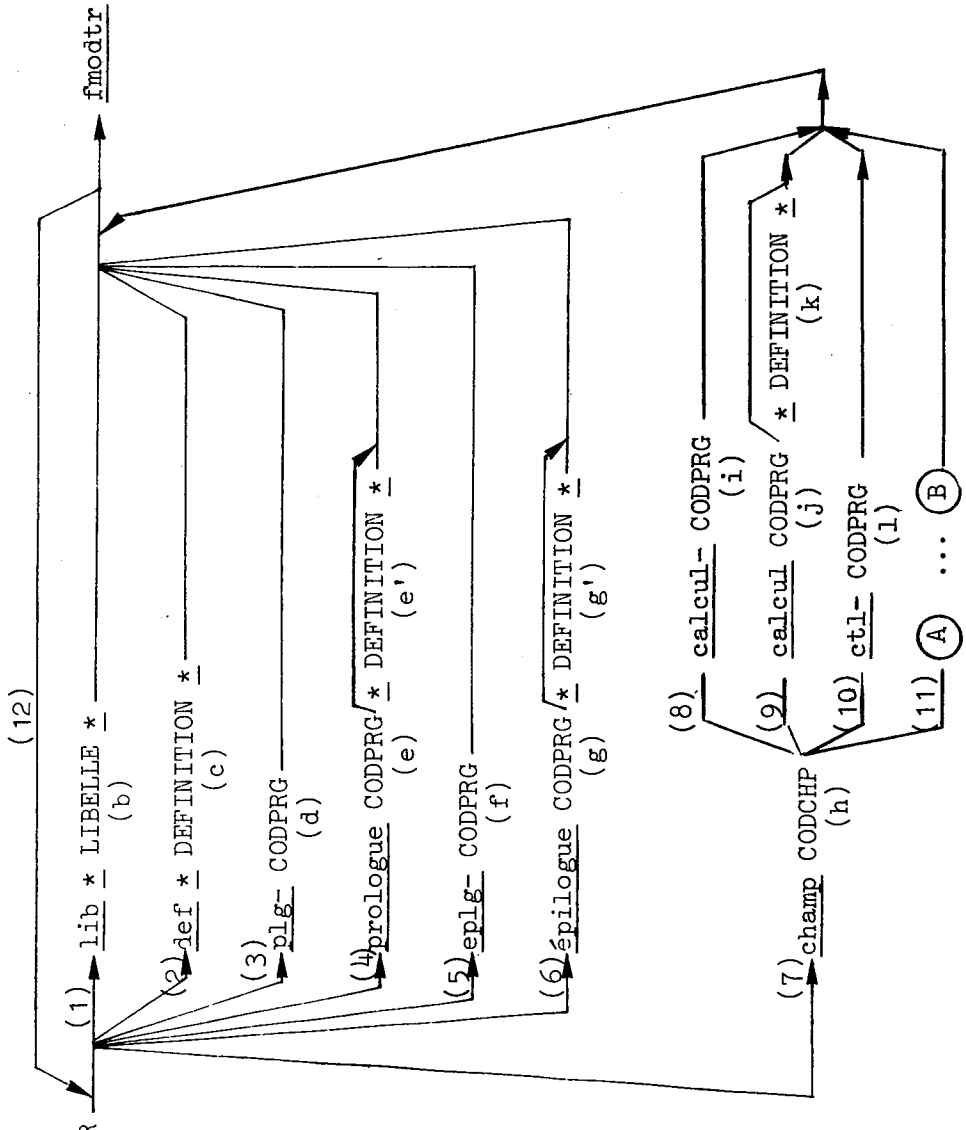
Soit le schéma de cette transaction :



III-3-4 - Instructions de modification et de suppression d'une transaction : modtr, suptr.

Instruction MODTR

action : modifier une ou plusieurs caractéristiques des spécifications d'une transaction d'entrée ou de sortie.


syntaxe :
modtr CODTR
(a)
(1)
lib * LIBELLE *
(b)
(2) def * DEFINITION *
(c)
(3) plg- CODPRG
(d)
(4) prologue CODPRG * DEFINITION *
(e)
(e')
(5) eplg- CODPRG
(f)
(6) épilogue CODPRG * DEFINITION *
(g)
(g')
(7) champ CODCHP
(h)
(8) calcul- CODPRG
(i)
(9) calcul CODPRG * DEFINITION *
(j)
(k)
(10) ctl- CODPRG
(l)
(11) (A) ... (B)
(12)
fmodtr
validité :

Zone	Repere	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
CODTR	(a)	1	R	- code pour identifier une transaction d'entrée ou de sortie déjà créée dans le modèle
/	(1)	/	/	
LIBELLE	(b)	9	N	- ces chemins permettent la spécification d'un nouveau libellé ou d'une nouvelle définition
/	(2)	/	/	de la transaction à modifier qui remplace l'ancienne spécification
DEFINITION	(c)	10	N	
/	(3)	/	/	- suppression du programme interne de code CODPRG du prologue de la transaction
CODPRG	(d)	1	R	- ce programme CODPRG n'est plus "attaché" au prologue de cette transaction, mais il n'est pas supprimé de la base et pourra faire l'objet d'un attachement ultérieur
/	(4)	/	/	- désignation ou déclaration d'un programme interne à attacher au prologue d'une transac-
CODPRG	(e)	1	R, N	tion d'entrée ou de sortie
DEFINITION	(e')	10	N	- ce chemin n'est possible que si la transaction CODTR (a) n'a encore aucun programme interne attaché à son prologue (sinon, il faut passer par le chemin (3) auparavant)
/	(5)	/	/	- suppression de l'épilogue de la transaction, mais le programme CODPRG est conservé dans
CODPRG	(f)	1	R	la base
/	(6)	/	/	- désignation ou déclaration d'un programme interne définissant l'épilogue de la
CODPRG	(g)	1	R, N	transaction
DEFINITION	(g')	10	N	- ce chemin ne peut être emprunté que si la transaction n'a encore aucun épilogue défini
/	(7)	/	/	- chemin utilisé pour modifier les caractéristiques relatives à un seul champ
CODCHP	(h)	1	R	- code d'un champ de la transaction de code CODTR (a)
/	(8)	/	/	- suppression de l'attachement du programme de calcul CODPRG (i) au champ
CODPRG	(i)	1	R	CODCHP (h)
/	(9)	/	/	- description d'un nouveau programme de calcul associé au champ CODCHP (h)
CODPRG	(j)	1	R, N	
DEFINITION	(k)	10	N	
/	(10)	/	/	- suppression de l'attachement du programme de contrôle CODPRG (l) au champ
CODPRG	(l)	1	R	CODCHP (h)
(A) ... (B)	(11)	/	/	- cf. p. III-55
				- définition de nouveaux programmes de contrôle pour le champ CODCHP (h)

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
/	(12)	/	/	- ce chemin autorise plusieurs modifications sur une même transaction avec la même instruction MODTR.

Notons que cette instruction ne permet pas une restructuration du graphe, mais simplement des changements de renseignements sur les noeuds (champs) de celui-ci.

Instruction SUPTR

action : supprimer une transaction d'entrée ou de sortie.

syntaxe : suptr CODTR fsuptr

validité : CODTR doit être le code d'une transaction d'entrée ou de sortie créée.

effets :

- suppression de toutes les spécifications relatives à une transaction, y compris le graphe des champs.
- mise à jour des listes de transactions (cf. § II-5-1) attachées aux ensembles et aux propriétés.
- conservation des déclarations des nouvelles propriétés qui ont été attachées définitivement à l'ensemble de référence.
- maintien des déclarations et descriptions éventuelles de programmes internes (contrôles particuliers, calculs, épilogue ou prologue) ; s'ils sont inutilisés par les transactions déjà créées ou sans intérêt pour les transactions futures à créer, ils seront supprimés à leur tour par une instruction de suppression de programmes internes (supprg).

III-4 - INSTRUCTIONS DE MISE AU POINT ET D'EVALUATION DES SPECIFICATIONS

Les instructions d'aide à l'analyse sont essentiellement des outils d'évaluation du modèle fonctionnel d'une application qui doivent contrôler la cohérence des descriptions fonctionnelles et en faciliter leur diffusion.

Ceci est réalisé dans MACSI-P par quatre types de programmes :

- les *programmes* de contrôle du modèle
- les *programmes de diagnostics* sur le modèle
- les *programmes d'édition* du modèle et de la documentation associée
- les *programmes de gestion des textes* documentaires.

L'utilisation de ces programmes est gérée par les trois instructions : CONTROLE, EDITION et MODEX.

III-4-1 - Contrôle des spécifications et diagnostics méthodologiques : instructions CONTROLE

- Les programmes de contrôle

Ces contrôles sont l'ensemble des opérations manuelles ou automatiques effectuées sur la base documentaire de l'application (modèle fonctionnel des données et des traitements de l'application) pour corriger des anomalies constatées ou bien pour compléter les descriptions.

On peut distinguer, dans les programmes de contrôle, deux catégories :

a) Les contrôles implicites : un grand nombre de ces contrôles ont déjà été évoqués précédemment ; ils sont liés directement à la méthodologie de description adoptée et ils assurent, d'une manière continue, la cohérence du modèle en cours d'élaboration.

Ces contrôles sont inhérents aux schémas des transactions MACSI-P et nous n'en rappelons que quelques-uns :

. respect de la syntaxe des schémas
. respect de la sémantique liée aux schémas (utilisation de codes discriminants, ...)
. respect d'un ordre logique strict dans l'utilisation des divers schémas
. respect des droits d'accès à un constituant (déclarer, créer, décrire, modifier ou supprimer)
. ...

b) Les contrôles explicites

Ces programmes nécessitent, pour être exécutés, un accès à l'ensemble de la base documentaire de l'application. Ils permettent essentiellement d'établir, à tout moment, un inventaire des spécifications incomplètes. Ils éditent les listes des composants "déclarés", et non encore "décrits" et des composants dont certaines caractéristiques sont encore indéfinies (ayant la valeur "U").

Trois programmes d'inventaires des spécifications incomplètes sont proposés :

CTL-ENS : établit la liste des ENSEMBLES (de base, relations, sous-ensembles ou agrégats) dont la DEFINITION ou la TAILLE sont encore indéfinies.

CTL-PRP : liste des PROPRIETES déclarées et non encore décrites ou dont la DEFINITION est indéfinie.

CTL-TRT : liste des TRANSACTIONS dont la définition n'est pas encore donnée.

Voir l'instruction CONTROLE dans les pages suivantes.

- Les programmes de diagnostics méthodologiques

Les programmes de contrôle présentés ci-dessus permettent uniquement de savoir, à tout moment, si la modélisation en cours de description est susceptible d'être complète : a-t-on complètement "décrit" (valorisé toutes les caractéristiques) tous les composants "déclarés" (ensembles, propriétés et transactions) ?

Aussi, le logiciel MACSI-P propose-t-il un autre ensemble de programmes, les programmes de diagnostics, pour aider l'analyste à apprécier la validité de la modélisation effectuée.

Ces programmes ne détectent pas des erreurs certaines (de non respect des règles MACSI-P), mais recensent les anomalies possibles : présomptions d'erreurs dans la modélisation. Ils exploitent, d'une part la base documentaire (modèle fonctionnel des données et des traitements de l'application), et d'autre part les inventaires gérés par le système MACSI-P, établissant des relations entre les données et les traitements.

Nous proposons trois groupes de programmes de diagnostics relatifs à :

- l'analyse du modèle fonctionnel des données de l'application
- l'analyse du modèle fonctionnel des traitements de l'application
- l'analyse des liens entre les données et les traitements de l'application.

Ces programmes sont notés DIA-MDn, DIA-MTn ou DIA-DTn suivant le groupe d'appartenance.

Donnons quelques-exemples de programmes de diagnostics.

DIA-MD1 : liste des ensembles X qui ne participent qu'à une seule relation (avec un autre ensemble Y)
intérêt : est-ce que X n'est pas une propriété de Y ? (cf. § II-2-2-4)

DIA-MD2 : liste des sous-ensembles qui n'ont aucune propriété particulière
intérêt : ces partitions ont-elles une utilité ?

DIA-MT1 : liste des transactions qui ont des prologues (épilogues) particuliers identiques
intérêt : ces prologues (épilogues) doivent-ils être exécutés au début (fin) de toutes ces transactions ?

* *DIA-DT1* : liste des ensembles qui ne sont utilisés que dans des transactions d'entrée

  intérêt : est-ce que la finalité de ce stockage de données est clairement analysée ?

* *DIA-DT2* : liste des ensembles ou propriétés n'apparaissant dans aucune transaction

  intérêt : est-ce qu'il était utile de les décrire dans le modèle fonctionnel des données ?
  N'y a-t-il pas encore des transactions à décrire ?

* *DIA-DT3* : liste des ensembles qui sont ensembles de référence de plusieurs transactions d'entrée avec la même opération envisagée (création, ou description, ou ...)

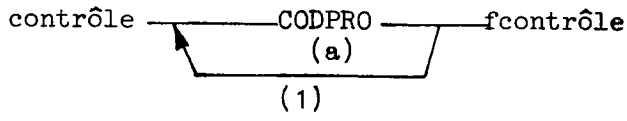
  intérêt : est-ce la même fonction décrite avec des modalités diverses ?
  Toutes ces transactions sont-elles utiles ?

L'exécution des programmes d'inventaire des spécifications incomplètes ou de diagnostics méthodologiques est demandée par l'instruction CONTROLE.

Instruction CONTROLE

action : demander l'exécution d'un ou plusieurs programmes d'inventaires de spécifications incomplètes ou de diagnostics méthodologiques.

syntaxe :



validité :

zone	repère	type	R/N	commentaires ou contrôles particuliers
CODPRO	(a)	5	R	- code d'un programme : - d'inventaire : CTL-ENS, CTL-PRP ou CTL-TRT ou - de diagnostic : DIA-MD1, DIA-MD2, ...
-	(1)	-	-	- si l'on désire faire exécuter encore un autre programme de contrôle

utilisation :

Après avoir donné de nombreuses spécifications fonctionnelles relatives à une application, il est souhaitable d'utiliser l'instruction CONTROLE suivante :

contrôle ctl-ens ctl-prp ctl-trt fcontrôle.

III-4-2 - Diffusion des spécifications : instruction EDITION

Dans MACSI-P, nous avons défini un ensemble de programmes documentaires pour restituer tout ou partie des descriptions et de la documentation d'un projet d'application.

Ces programmes sont utilisés pour aider l'analyste à continuer la modélisation de l'application ou pour changer d'étape.

Nous distinguerons quatre types de programmes documentaires :

a) édition d'un dictionnaire : il est important pour un chef de projet d'éditer à intervalles réguliers la liste des codes et libellés des constituants déclarés.
Ce dictionnaire permet aux analystes de mieux choisir les codes des nouveaux constituants.

b) édition récapitulative des spécifications d'un constituant : il est nécessaire d'avoir la possibilité d'éditer toutes les spécifications relatives à un constituant, pour les compléter ou pour évaluer les conséquences des modifications.

c) dossier partiel permettant de récapituler les spécifications utiles à une tâche d'analyse précise. Par exemple, il est nécessaire d'éditer la liste de tous les ensembles de base, relations, sous-ensembles, agrégats et leurs propriétés pour préparer l'étape de modélisation des traitements.

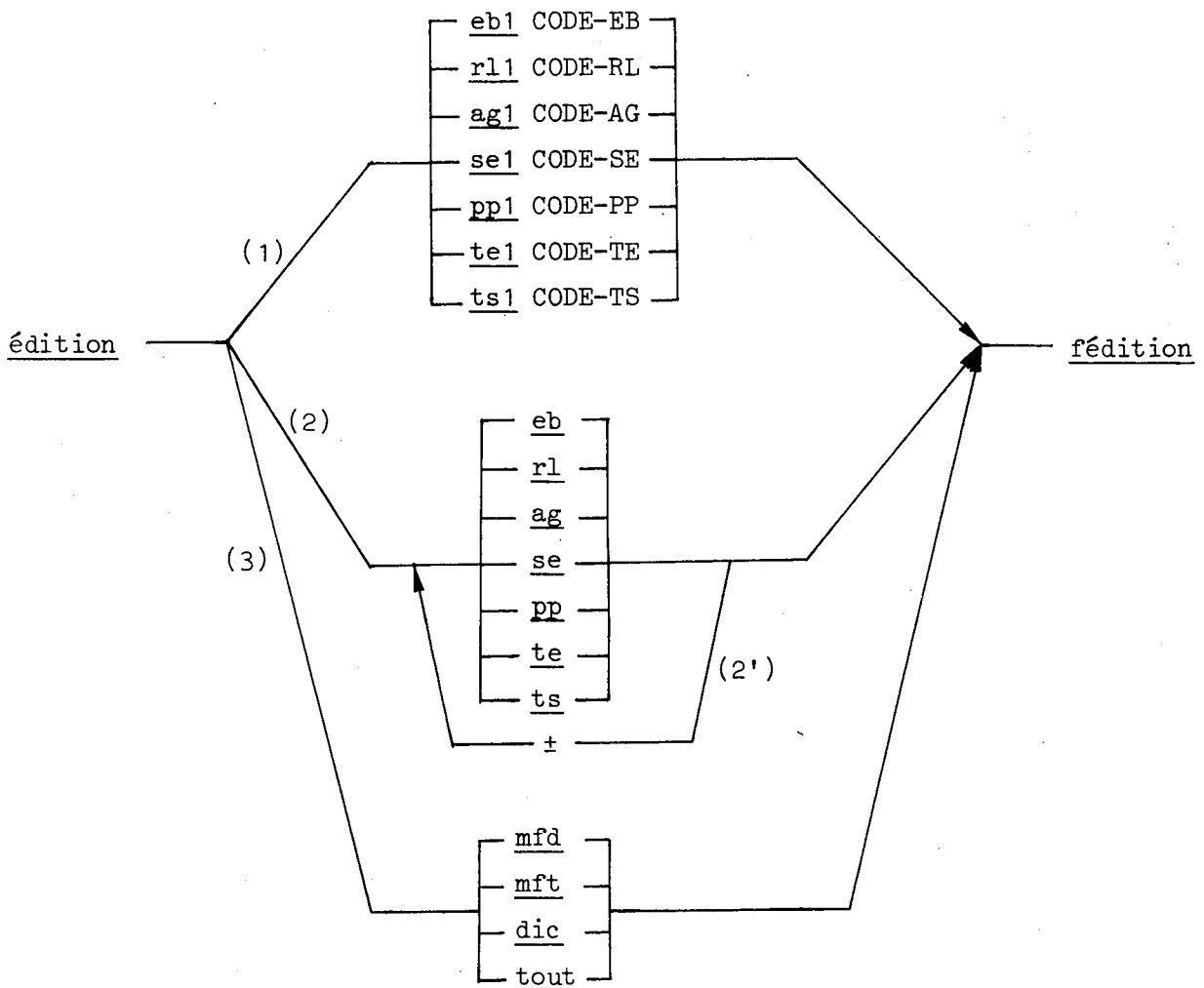
d) dossier exhaustif : lorsque l'analyse d'un projet est considérée comme terminée (l'instruction CONTROLE ne signale aucune anomalie), le chef de projet peut alors procéder à une édition complète du fichier documentaire du projet.

L'exécution de ces différents programmes documentaires est assurée par l'instruction EDITION.

Instruction EDITION

action : éditer tout ou partie des spécifications d'un projet

syntaxe :



validité : attention, les chemins (2) et (3) ne sont utilisables que lors d'une session batch.

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
/	(1)	/	/	- ce chemin est emprunté dans le cas d'une édition concernant toutes les spécifications d'un seul constituant du modèle fonctionnel de l'application
				- cas de :
eb1	/	0	/	un ensemble de base
CODE-EB	/	1	R	
rl1	/	0	/	une relation
CODE-RL	/	1	R	
ag1	/	0	/	un agrégat
CODE-AG	/	1	R	
se1	/	0	/	un sous-ensemble
CODE-SE	/	1	R	
pp1	/	0	/	une propriété
CODE-PP	/	1	R	
te1	/	0	/	une transaction d'entrée
CODE-TE	/	1	R	
ts1	/	0	/	une transaction de sortie
CODE-TS	/	1	R	
/	(2)	/	/	- cas d'un dossier partiel regroupant les spécifications de tous :
eb	/	0	/	- les ensembles de base
rl	/	0	/	- les relations
ag	/	0	/	- les agrégats
se	/	0	/	- les sous-ensembles
pp	/	0	/	- les propriétés
te	/	0	/	- les transactions d'entrée
ts	/	0	/	- les transactions de sortie
±	(2')	0	/	- si l'on veut enchaîner plusieurs dossiers partiels
mfd	/	0	/	- permet l'édition du modèle fonctionnel des données d'une application
				- c'est la séquence prédéterminée eb + rl + ag + se + pp

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
mft	/	0	/	- permet l'édition du modèle fonctionnel des traitements d'une application - mft = te + ts
tout	/	0	/	- permet l'édition de tout le modèle fonctionnel d'une application - tout = mfd + mft
dic	/	0	/	- édition du dictionnaire de l'application

utilisations :

Un analyste peut écrire l'instruction suivante :

édition eb1 SALLE fédition

pour obtenir la restitution des spécifications saisies
ou bien :

édition eb + pp fédition

avant de décrire des relations, sous-ensembles ou agrégats..

Mais il fera

édition mfd fédition

avant d'utiliser les instructions CRETR-E et CRETR-S.

Le chef de projet demandera :

édition tout fédition

lorsqu'il jugera que les spécifications fonctionnelles sont achevées.

III-4-3 - Gestion des textes : instruction MODEX

Tout constituant (ensemble de base, relation, propriété, ... transaction ou programme ...) du modèle fonctionnel proposé possède une caractéristique sous forme de texte libre en langue naturelle que nous avons appelé DEFINITION dans les différentes instructions du langage MACSI-P.

Un texte est représenté par une suite de 200 lignes au plus et de 60 caractères maximum par ligne.

L'analyste peut remplacer le texte de définition d'un constituant par un nouveau texte jugé plus satisfaisant en utilisant des instructions MACSI-P : MODENS, MODPP ou MODTR suivant la nature du constituant. Ces modifications, possibles dans le cas d'un texte court de quelques lignes, deviennent longues et sources d'erreurs pour un texte de plusieurs dizaines de lignes. Elles sont fastidieuses si l'analyste ne désire modifier qu'une petite partie du texte initial : un éditeur de textes est nécessaire.

Le système Socrate que nous utilisons possède son propre éditeur de textes qui est placé au même niveau que les différents processeurs du système. Or, l'accès à notre système MACSI-P n'est possible qu'à partir du processeur de requêtes SOCRATE : l'éditeur Socrate n'est donc pas à la disposition de l'analyste qui travaille dans l'environnement MACSI-P.

Pour cette raison, nous avons défini un éditeur de textes sous MACSI-P : l'instruction MODEX.

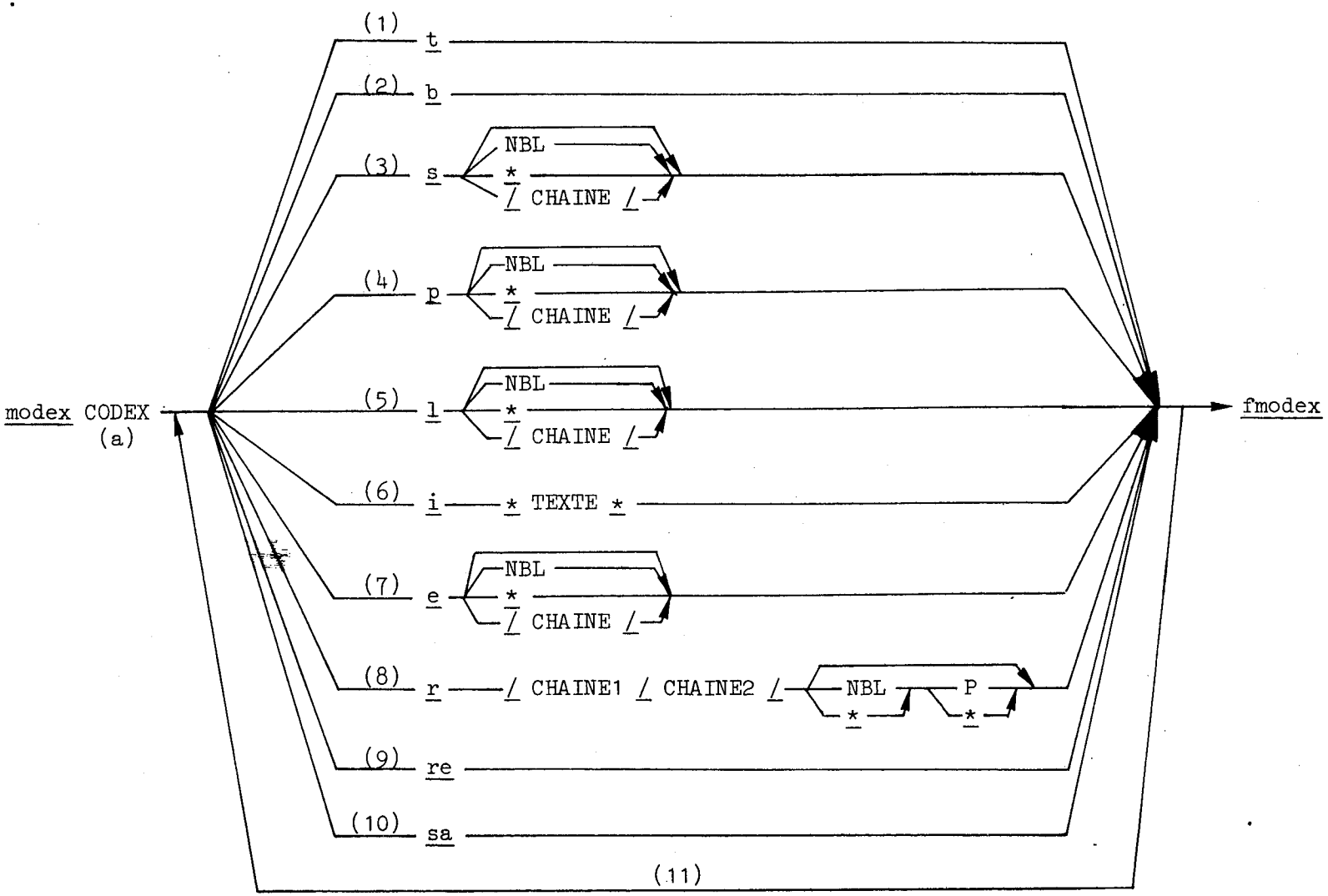
Nous avons regroupé tous les textes de définition des constituants dans une zone spéciale appelée EXPLICATION (entité EXPLIC de la structure Socrate documentaire de MACSI-P). Chaque texte (explication) possède un code d'identification numérique (CODEX) géré par MACSI-P et communiqué aux analystes ; ce code n'est pas donné par l'auteur du texte.

L'instruction MODEX est définie sur ces EXPLICATIONS.

Instruction MODEX : un éditeur de texte

action : modifier le contenu d'une explication (texte de définition d'un constituant)

syntaxe :



validité :

- cette instruction doit être utilisée de préférence en conversationnel.
- le texte à modifier est dupliqué dans une zone de travail appelée SAVEX ; un pointeur permet de repérer la "ligne courante" du texte dans SAVEX. Au cours de cette recopie, une ligne vide (LV1) de début de texte et une ligne vide (LV2) de fin de texte sont concaténées aux extrémités du texte. Le pointeur de ligne courante est alors placé sur la première ligne réelle au début du texte (sauf si le texte initial est vide, auquel cas le pointeur est positionné sur la ligne vide LV1).
- le texte initial reste inchangé pendant les manipulations qui se font sur cette zone de travail ; seul l'utilisateur décidera du remplacement du texte initial par le texte transformé avec la commande SA(UVER).
- l'instruction modex est composée de trois types de commande :
  - des commandes de déplacement du pointeur de la ligne courante (t, b, s, p)
  - des commandes d'impression ou de modification (l, i, e, r)
  - et des commandes de sauvegarde ou d'annulation des modifications (sa, re).

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
CODEX	(a)	1	R	- numéro d'une explication dont le texte est à modifier
t	(1)	0	/	- t(ête) : . placer le pointeur au début du texte avant la première ligne réelle (sur LV1) . le message '... EN TETE ...' est imprimé
b	(2)	0	/	- b(as) : . placer le pointeur sur la dernière ligne réelle du texte . la ligne courante est imprimée
s	(3)	0	/	- s(uivant) : . déplacer le pointeur de ligne courante vers le bas du texte
NBL	/	3	N	- s : déplacement d'une ligne
*	/	0	/	- s NBL : : déplacement de NBL lignes (NBL ≥ 1)
CHAINE	/	9	N	- s * : déplacement sur la dernière ligne réelle du texte - s /CHAINE/ : déplacement jusqu'à la première ligne contenant la chaîne de caractère = CHAINE . après le déplacement, la ligne courante est imprimée . si le nombre de lignes à sauter fait que le déplacement dépasse la dernière ligne réelle du texte, alors le message '... EN BAS ...' est imprimé, et le pointeur reste positionné sur LV2.
p	(4)	0	/	- p(récédent) : . déplacer le pointeur de ligne courante vers le haut du texte
NBL	/	3	N	- p : déplacement d'une ligne
*	/	0	/	- p NBL : déplacement de NBL lignes (NBL ≥ 1)
CHAINE	/	9	N	- p * : déplacement sur la première ligne réelle du texte - p /CHAINE/ : déplacement jusqu'à la première ligne contenant la chaîne de caractères CHAINE . après le déplacement, la ligne courante est imprimée . si le nombre de lignes à ignorer fait que le déplacement dépasse la première ligne réelle du texte, alors le message '... EN TETE ...' est imprimé, et le pointeur reste positionné sur LV1

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
l	(5)	0	/	- l(ister) : . imprimer une ou plusieurs lignes du texte, ligne courante comprise, en se déplaçant vers le bas
NBL	/	3	N	. la ligne courante devient la dernière ligne imprimée
*	/	0	/	
CHAINE	/	9	N	- l : imprimer la ligne courante - l NBL : imprimer NBL lignes à partir de la ligne courante (NBL ≥ 1) - l * : imprimer toutes les lignes à partir de la ligne courante jusqu'à la fin du texte - l /CHAINE/ : imprimer toutes les lignes à partir de la ligne courante jusqu'à la première ligne contenant CHAINE . si le déplacement dépasse la dernière ligne réelle du texte, alors le message '... en BAS ...' est imprimé et le pointeur est positionné sur LV2
i	(6)	0	/	- i(nsérer) : . insérer une ou plusieurs lignes de 60 caractères entre la ligne courante et la ligne suivante
TEXTE	/	10	N	. le pointeur sera placé sur la dernière ligne insérée
e	(7)	0	/	- e(ffacer) : . effacer une ou plusieurs lignes, ligne courante comprise, en se déplaçant vers le bas
NBL	/	3	N	
*	/	0	/	- e : effacer la ligne courante
CHAINE	/	9	N	- e NBL : effacer NBL lignes à partir de la ligne courante (NBL ≥ 1) - e * : effacer toutes les lignes à partir de la ligne courante jusqu'à la fin du texte - e /CHAINE/ : effacer toutes les lignes à partir de la ligne courante jusqu'à la première ligne, contenant CHAINE, comprise . le pointeur sera placé sur la ligne suivant la dernière ligne supprimée . imprime la nouvelle ligne courante . si le déplacement dépasse la dernière ligne réelle du texte, alors le message '... en BAS ...' est imprimé et le pointeur est positionné sur LV2

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
r	(8)	0	/	- r(emplacer) : . remplacer une ou P occurrences de CHAINE1 par CHAINE2 sur une ou NBL lignes à partir de la ligne courante comprise
CHAINE1	/	9	N	- r /CHAINE1/CHAINE2/ : remplace dans la ligne courante la première occurrence de CHAINE1 par CHAINE2
CHAINE2	/	9	N	- r /CHAINE1/CHAINE2/ * * : le remplacement se fait sur toutes les occurrences de toutes les lignes jusqu'à la fin du texte
*	/	0	/	- r /CHAINE1/CHAINE2/ * P : le remplacement se fait sur les P (≥ 1) premières occurrences dans chaque ligne jusqu'à la fin du texte
NBL	/	3	N	- r /CHAINE1/CHAINE2/ NBL * : le remplacement se fait sur toutes les occurrences dans NBL (≥ 1) lignes
P	/	3	N	- r /CHAINE1/CHAINE2/ NBL P : le changement s'effectue sur les P (≥ 1) premières occurrences dans NBL (≥ 1) lignes . le pointeur est alors placé sur la dernière ligne modifiée . les lignes modifiées sont imprimées
re	(9)	0	/	- re(staurer) : permet l'annulation de toutes les manipulations qui ont été faites depuis la dernière utilisation de la commande sauver ou de l'entrée dans l'éditeur si aucun sauver n'a été fait. La zone de travail SAVEX est de nouveau chargée avec le texte initial
sa	(10)	0	/	- sa(uver) : permet la sauvegarde des transformations faites. . le texte initial est remplacé par le texte contenu dans la zone SAVEX
/	(11)	/	/	- pour décrire plusieurs commandes de l'éditeur
fmodex	/	0	/	- fin de l'instruction modex - le nouveau texte (contenu dans la zone de travail SAVEX) remplace le texte initial qui n'est plus disponible.

CHAPITRE IV

UN OUTIL : LE LOGICIEL MACSI-P

IV-1 - Spécifications générales du logiciel MACSI-P

IV-2 - Etape 1 : Initialisation du logiciel MACSI-P

IV- 3 -Etape 2 : Elaboration de la structure fonctionnelle d'une application

IV-4 - Etape 3 : Réalisation d'un prototype

IV-5 - Etape 4 : Utilisation d'un prototype.

IV-1 - SPECIFICATIONS GENERALES DU LOGICIEL MACSI-P

Le logiciel dont nous présentons les spécifications a été conçu et réalisé pour servir de support à notre méthode d'aide à la conception des systèmes d'informations.

Le but de ce chapitre est de donner essentiellement

- les *règles d'utilisation* du logiciel MACSI-P,
- et - les *principes de base* de sa conception,

concernant l'analyse fonctionnelle d'une application et la construction d'un prototype de cette application.

La réalisation technique de l'outil est décrite dans l'annexe C.

Il apparaît à la lumière de la démarche proposée précédemment (chapitres II et III) que la conception d'un système d'informations, centré ou non autour d'une base de données, nécessite une analyse minutieuse. Pour ce faire, il est indispensable de saisir méthodologiquement les renseignements obtenus tout au long de l'analyse et d'en effectuer la synthèse. De plus, à l'issue d'une telle analyse fonctionnelle, nous voulons réaliser quasi automatiquement un prototype de l'application et disposer de fonctions d'aide et de contrôle de son utilisation.

Aussi, nous a-t-il semblé intéressant de mettre à la disposition des concepteurs un "outil informatique" susceptible de les aider dans ce travail et dont l'utilisation doit respecter des règles méthodologiques strictes.

IV-1-1 - Processus logique de mise en oeuvre du logiciel MACSI-P en quatre étapes

Après avoir décrit dans les chapitres précédents les différents constituants du modèle de représentation d'une application informatique, il est nécessaire maintenant de préciser l'organisation générale qui doit être adoptée dans la mise en oeuvre du logiciel MACSI-P.

Cette mise en oeuvre se déroule nécessairement suivant quatre étapes distinctes schématisées dans la figure suivante :

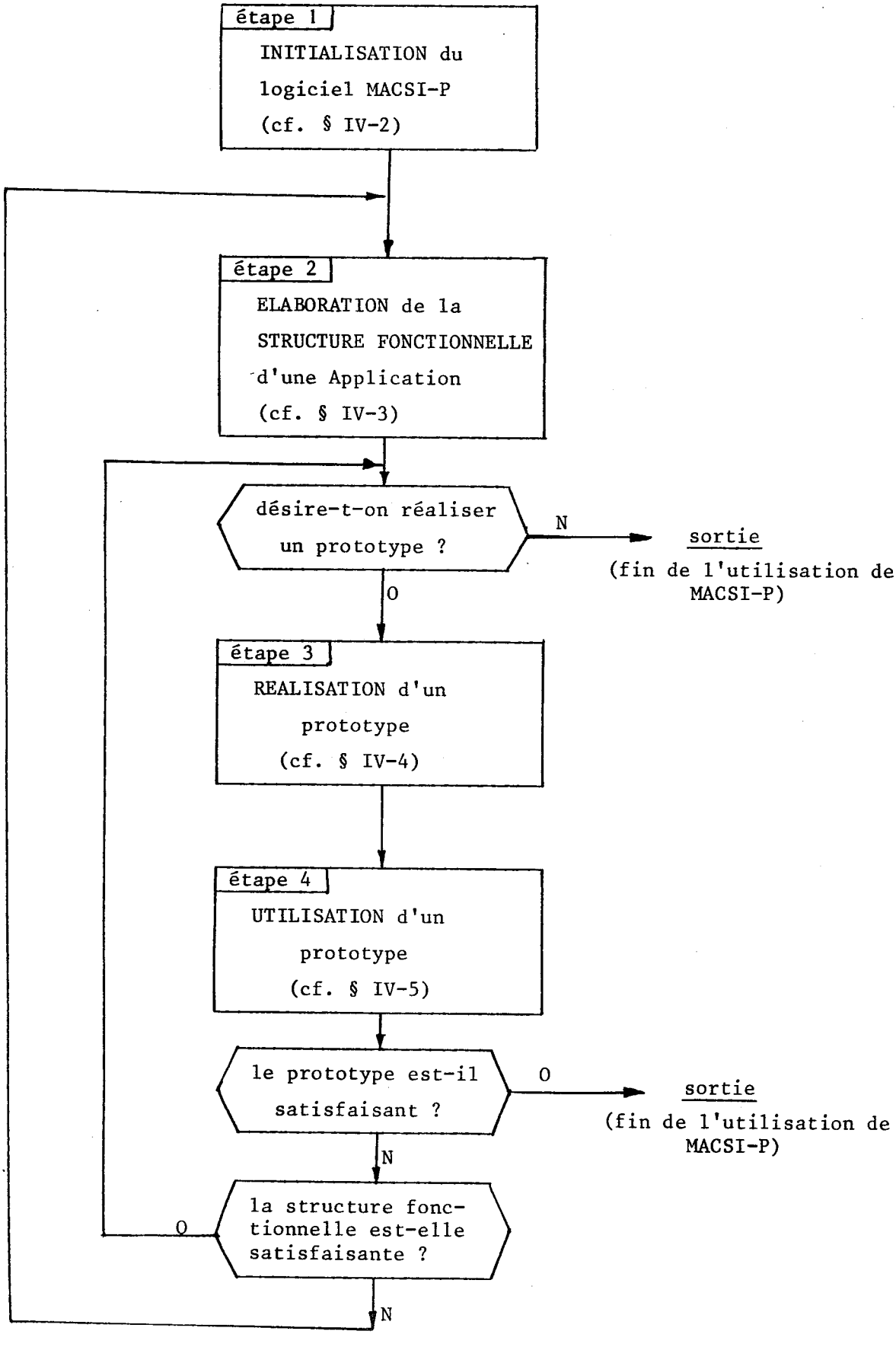


Figure : Processus logique d'utilisation du logiciel MACSI-P

Etape 1 : l'initialisation du logiciel MACSI-P consiste à créer une base de données SOCRATE contenant
d'une part le logiciel MACSI-P
et d'autre part l'initialisation d'un projet d'application.

Etape 2 : c'est la description sémantique des données et des traitements d'une application informatique de gestion en vue de constituer son modèle fonctionnel.

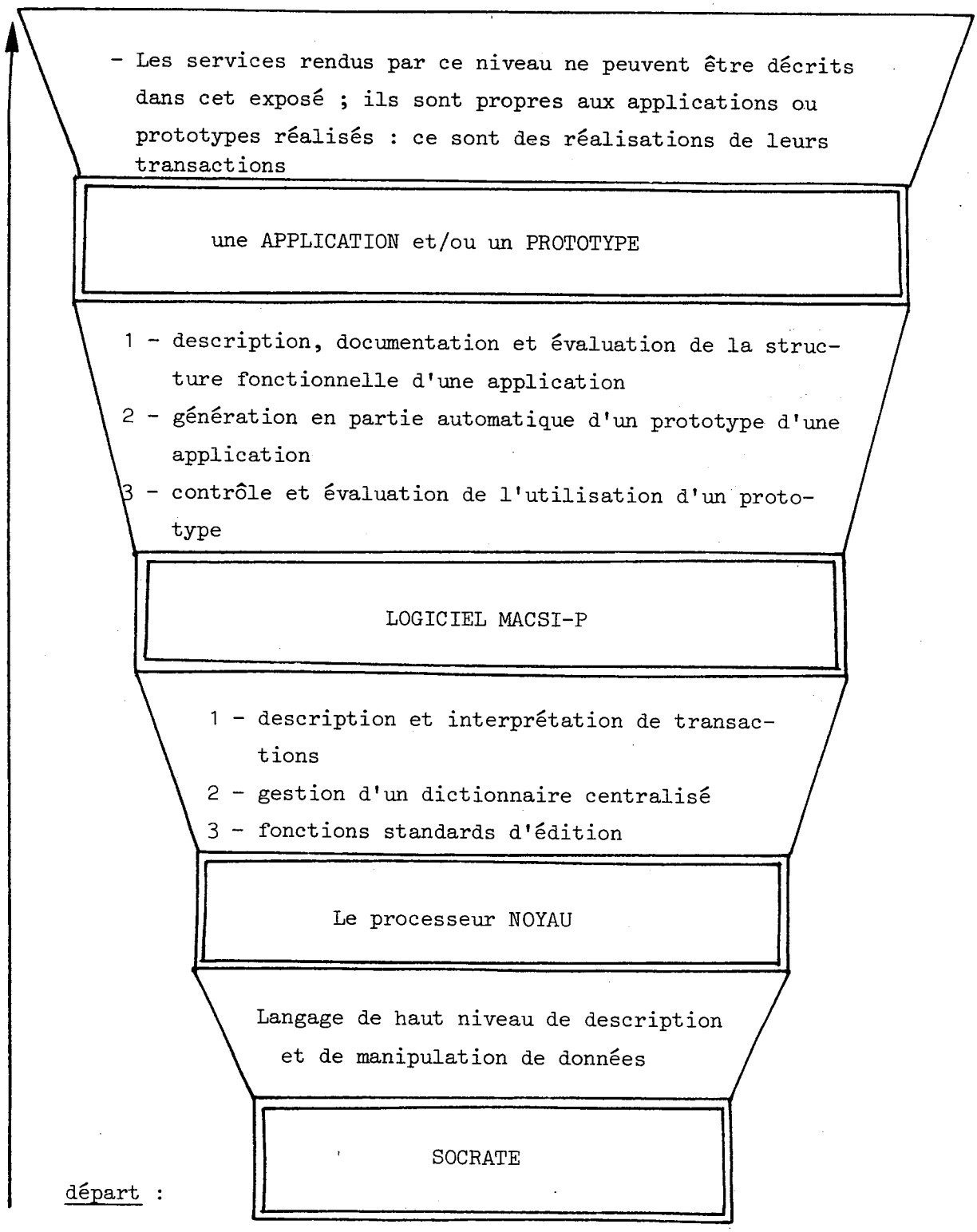
Etape 3 : à partir du dossier décrit au cours de l'étape 2, MACSI-P propose une réalisation quasi automatique d'un prototype de l'application spécifiée.

Etape 4 : le prototype construit dans l'étape 3 est alors livré aux futurs utilisateurs de l'application : leurs remarques permettent de valider ou de modifier le prototype avant de passer à la réalisation de la version finale de l'application.

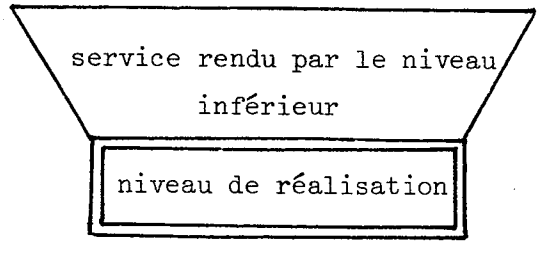
IV-1-2 - Principes élémentaires de réalisation

Pour atteindre le but visé (réalisation et utilisation d'un prototype d'une application), nous avons procédé à des réalisations par niveaux successifs. Le niveau de départ est le système SOCRATE de gestion de bases de données ; celui d'arrivée est une base de données prototype. Chaque étape représente un logiciel qui offre un ensemble de services en termes de fonctio à ses utilisateurs potentiels et qui, éventuellement, autorise l'accès à des services des niveaux inférieurs.

Nous représentons cette hiérarchie de niveaux et les fonctions assurées dans la figure ci-après que nous détaillons dans la suite de l'exposé.


- Les services rendus par ce niveau ne peuvent être décrits dans cet exposé ; ils sont propres aux applications ou prototypes réalisés : ce sont des réalisations de leurs transactions
une APPLICATION et/ou un PROTOTYPE
1 - description, documentation et évaluation de la structure fonctionnelle d'une application
2 - génération en partie automatique d'un prototype d'une application
3 - contrôle et évaluation de l'utilisation d'un prototype
LOGICIEL MACSI-P
1 - description et interprétation de transactions
2 - gestion d'un dictionnaire centralisé
3 - fonctions standards d'édition
Le processeur NOYAU
Langage de haut niveau de description et de manipulation de données
SOCRATE
départ :

légende :


service rendu par le niveau inférieur
niveau de réalisation

a) Fonctions du logiciel MACSI-P

Nous n'avons pas limité les fonctions de cet outil aux seuls besoins des concepteurs. Aussi, classons-nous ces fonctions en trois rubriques.

Fonctions destinées à satisfaire les besoins des concepteurs

A ce titre, le logiciel réalise deux catégories de fonctions :

- fonctions de description et de documentation d'applications informatiques, par normalisation et saisie des descriptions documentées ;
- fonction de synthèse pour fournir les éléments indispensables à l'élaboration d'une structure fonctionnelle par :
  - restitution complète ou sélective des descriptions et d la documentation,
  - utilisation de programmes de diagnostics méthodologiques.

Fonctions destinées à satisfaire les besoins des analystes-programmeurs qui ont la charge de réaliser un prototype d'une application :

- fonction documentaire destinée à fournir les divers renseignements nécessaires pour mener à bien les tâches de programmation et de mise au point ;
- fonction de traduction d'une structure fonctionnelle de données en une structure d'accès et de stockage, ou de schémas de transaction en automates interprétables.

Fonctions relatives aux besoins des utilisateurs du prototype

- fonction documentaire : l'outil doit leur fournir le dictionnaire du projet et le descriptif des transactions disponibles ;
- fonction d'exécution et d'utilisation du prototype.

Pour utiliser une de ces fonctions, il faut écrire une REALISATION D'UNE TRANSACTION du langage MACSI-P (chapitre III) ou de l'application.

Notons que les transactions d'une application sont décrites avec le même formalisme que les transactions du langage MACSI-P : le principe directeur du projet MACSI-P repose sur la nécessité de disposer d'un formalisme et

d'une méthodologie uniques pour :

- l'analyse,
- la description,
- la documentation,
- la mise en oeuvre d'une application.

Remarques :

L'ensemble des schémas possibles des transactions du langage MACSI-P définit notre méthodologie de conception d'une application informatique.

L'ensemble des schémas des transactions d'une application constitue sa méthodologie de mise en oeuvre.

Ces méthodologies doivent être évolutives et adaptables à de nouveaux besoins, ceci dans des délais de programmation très courts. Il était donc nécessaire de disposer d'un ensemble d'outils informatiques utilisables pour construire, exploiter et modifier les méthodes proposées.

Or, dans le cadre de travaux développés à l'IFC*, de tels outils ont été réalisés sous la forme d'une *famille de sous-programmes* Socrate appelée NOYAU.

b) Services fournis par le système "NOYAU"

"NOYAU" a été conçu pour faciliter la mise au point, sous Socrate, d'applications transactionnelles. Les performances de ce système sont faibles, mais il offre une grande souplesse, tant dans la description que dans l'exploitation de telles applications. Après avoir réalisé quelques extensions, nous l'avons utilisé pour développer cette première version du logiciel MACSI-P.

Dans tout ce chapitre, "NOYAU" dénote cette famille de sous-programmes Socrate. Dans un dossier technique, nous décrivons les différentes fonctions assurées par "NOYAU" (cf. annexe C).

---

*I.F.C. : *Institut de Formation et de Conseil - Place Doyen Gosse - 38031 GRENOBLE Cedex.*

Néanmoins nous pouvons résumer ici ces fonctions :
- description documentée de transactions sous forme de graphes,
- validation de transactions en assurant les lectures et les contrôles,
- gestion de l'enchaînement de transactions,
- constitution d'un dictionnaire centralisé des données,
- définition d'un dossier standard de sortie,
- utilisation d'un moniteur principal fonctionnant indifféremment en traitement conversationnel ou par lots et offrant des procédures d'apprentissage en conversationnel.

c) Choix du système SOCRATE

Socrate offre tous les services attendus d'un système de gestion de bases de données. Il présente l'avantage de posséder un langage de description et de manipulation de données extrêmement synthétique et de haut niveau utilisable en traitement conversationnel ou par lots.

D'autre part, les instructions de traitement offrent trois possibilités :

- succession linéaire d'instructions de créations (G), de mises à jour (M) et d'interrogations (I).
- traitement par balayage : pour
  ...
  fin
- traitement répétitif contrôlé : faire
  ...
  refaire
  fin

Cette réduction apparente des possibilités (en particulier l'absence d'instructions de branchement) facilite la compréhension de la logique adoptée po un programme qui est naturellement structuré : le schéma du programme apparaît immédiatement à l'étude des différentes imbrications des procédures de balayages et de traitements répétitifs.

IV-1-3 - Fonctionnement de MACSI-P : les instructions MACSI et MACSIBATCH, les fonctions .SOS, .AIDE, .FAID, .EXPL, .EDIT et .STOP

Le logiciel MACSI-P fonctionne sous deux modes :

- en *conversationnel*, à partir d'un terminal interactif (machine à écrire, écran) ;
- en *traitement par lots*.

A chacun de ces modes correspond un accès particulier à la base MACSI-P au moyen des instructions MACSI et MACSIBATCH (cf. p. IV-13), mais quel que soit le mode de travail choisi, l'analyste fournit les mêmes réalisations de transactions, avec le même formalisme.

a) Fonctionnement en traitement par lots

L'écriture des instructions MACSI-P se fait sur papier ordinaire en format libre. Ces papiers sont perforés à raison d'une carte par ligne manuscrite.

La première ligne doit être : *macsibatch CODPRO CODANAL ?*

...

La dernière : *fin*

Il faut demander au responsable du fonctionnement de MACSI-P les paquets de cartes à placer devant et derrière celui issu de l'écriture d'une description. Il suffit de faire lire le paquet ainsi constitué par un lecteur de cartes pour analyser et enregistrer les spécifications relatives à un projet.

Ce mode de fonctionnement se caractérise par :

- la possibilité de traiter de gros volumes de description (plusieurs centaines de lignes manuscrites) ;
- l'impossibilité de corriger immédiatement les erreurs, la production d'une liste d'anomalies et l'abandon des instructions erronées et la nécessité de contrôler une instruction et de la lancer en machine autant de fois qu'elle est erronée ;

- la possibilité de sortir une documentation (instruction EDITION, cf. § III.4) parfois volumineuse à l'imprimante.

b) Fonctionnement en conversationnel

Le fonctionnement en conversationnel s'utilise plus particulièrement dans deux cas :

- pour apprendre les instructions de MACSI-P ; on utilise alors les fonctions d'assistance ;
- pour entrer des descriptions de faible volume, car dès que le volume devient important, le traitement par lots est préférable ; on ne peut guère envisager de rentrer à partir d'un terminal plus de 100 lignes manuscrites à l'heure.

Il se caractérise par :

- une détection d'erreurs et une demande de correction instantanée
- une possibilité d'interruption laissée à l'initiative de la personne qui utilise le terminal (fonction .STOP) ;
- peu de possibilités d'interrogation des descriptions et de la documentation déjà enregistrées dans la base documentaire.

Il se déroule de la façon suivante :

- l'*ordinateur* imprime "-" sur le terminal et se met en attente.
- l'*analyste* frappe une ligne, en format libre en séparant tous les mots par un espace, y compris les "*" et appuie sur la touche "RETURN" pour expédier cette ligne à l'ordinateur.
- l'*ordinateur* analyse la ligne de gauche à droite, en s'arrêtant à chaque espace. Chaque mot est comparé au schéma en cours d'exécution afin de :
  - . vérifier son occurrence s'il s'agit d'un mot-clé,
  - . interpréter son type s'il s'agit d'une valeur.
- *s'il y a une erreur*, l'ordinateur émet le message : [ERREUR nn sur MOT xxxxxx libellé de l'erreur], qui signifie que l'erreur de numéro nn a été détectée sur le mot xxxxxx dans la ligne qui vient d'être frappée.

- l'*ordinateur* réimprime un "-" en début de ligne sur le terminal.
- l'*analyste* frappe une nouvelle ligne à partir du mot qui a causé l'erreur ; tout ce qui se trouvait à droite du mot erroné sur la ligne précédente est perdu.
- au cas où il y a absence ou incorrection orthographique d'un mot-clé, l'ordinateur émet le message :
  ABSENCE D'UN MOT-CLE

Exemple de mise en oeuvre à partir d'un terminal :

Après avoir mis le terminal sous tension (touche ON ou LINE), appuyer sur la touche BREAK.
Le dialogue suivant s'établit :

Terminal	*Réponses de l'analyste*
SOCRATE A VOTRE SERVICE	
$	*LOGIN*
ACC. NUMBER :	*0072*
NAME :	*PAOLI*
BASE :	*MAC-P*
MOT DE PASSE :	*MAC-P*
RIEN A SIGNALER	
$	*GØ Ø*
QUESTION :	*macsi codepro codeanal ?*
-	...
	(frappe de diverses instructions de description)
-	...
-	*FIN*
/// FIN DE SESSION	
QUESTION :	*NØN*
$	*LØGØUT*
LØGØUT LE : JJ/MM/AA	
A HH.MM.SS	

c) Fonctions d'assistance pédagogique en conversationnel

Pour aider l'analyste, des fonctions d'assistance pédagogique ont été définies. Elles se caractérisent par "." (point) en premier caractère. Toute suite de caractères comprise entre deux espaces et commençant par un "." est interprétée comme l'appel d'une fonction particulière. Si l'ordinateur ne la connaît pas, il émet le message :

FONCTION .XXXX INCONNUE

Une fonction d'assistance est donc un mot réservé qui peut être inséré à n'importe quel endroit d'une réalisation d'une transaction MACSI-P, excepté à l'intérieur d'un texte, d'une ligne ou d'un mot.

Notons que ces fonctions sont définies *au niveau du processeur* NOYAU, mais peuvent être utilisées par les deux niveaux supérieurs : le logiciel MACSI-P et le prototype d'une application. Nous les avons décrites complètement (§ C-6-6) et nous les résumons dans le tableau suivant.

fonction	définition
.SOS	indiquer à l'utilisateur ce qu'il est autorisé à faire à un instant donné
.AIDE .FAID	.AIDE permet un appel continu à la fonction .SOS ; cette aide permanente peut être annulée par la fonction .FAID
.EXPL	donner des commentaires explicatifs plus complets que ceux fournis par la fonction .SOS
.EDIT	édition à un instant donné de tous les mots lus et analysés depuis le début de la validation de la réalisation d'une transaction en cours
.STOP	abandonner la réalisation en cours de transmissions et de validation pour recommencer une autre réalisation.

Exemple :

- *Un utilisateur est en cours de description d'une propriété (transaction decpp) :*

  ... mode 1 type 3 | *.SOS* |

- *L'ordinateur imprime :*

  DONNER MOT CLE PLAGE : SI LA PROPRIETE EST DE TYPE 3 ou 4

  DONNER MOT CLE LISTE : SI LA PROPRIETE EST DE TYPE 5, 6, 7 ou 8

  DONNER MOT CLE COMP : SI LA PROPRIETE EST COMPOSEE ET DE TYPE

1, 2, 9 ou 10

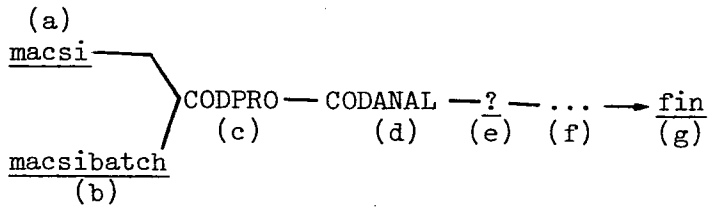
DONNER MOT CLE FDECPP : SI LA PROPRIETE EST ELEMENTAIRE ET DE TYPE 1, 2, 9 ou 10

- *L'utilisateur indique alors :*
  - *plage I-NBPERS* [.STOP]
- *La description de la propriété est alors abandonnée ; cette propriété reste néanmoins déclarée, mais non décrite. Il faudra refaire une instruction decpp sur cette propriété.*

d) Instructions MACSI et MACSIBATCH

action : *démarrer une session* de travail sur un projet en mode conversationnel ou par lots. Ce sont des instructions d'accès à un projet. Une session concerne un "projet" pour lequel "un analyste" fournit des descriptions.

syntaxe :



validité : (voir page suivante)

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
macsi	(a)	0	/	- début d'une session MACSI en conversationnel
macsibatch	(b)	0	/	- début d'une session MACSI en traitement par lots - une session en batch est constituée d'un paquet de cartes ou d'une saisie sur bande magnétique. Le premier enregistrement doit commencer par MACSIBATCH, le dernier ne contiendra que fin (les enregistrements doivent faire 80 caractères)
CODPRO	(c)	1	R	- code d'un projet déjà déclaré
CODANAL	(d)	1	R	- code d'un analyste déjà déclaré et affecté au projet CODPRO (c)
?	(e)	0	/	- marque la fin de l'accès à un projet et le début de la session proprement dite
...	(f)	/	/	- liste d'instructions MACSI-P
fin	(g)	0	/	- fin d'une session MACSI-P

Note : La définition de ces tableaux est décrite au début du chapitre III :

- les deux premières colonnes indiquent le "nom" etle "repère" d'une zone du schéma,
- la troisième colonne précise le "type" de la valeur à lire (0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ou 10),
- dans la quatrième colonne, R = valeur à Rechercher et N = valeur Nouvelle,
- la cinquième colonne commente le schéma.

Utilisation :

```
macsibatch GESTIONIUT PAOLI ?
        creens SALLE * salles de cours de l'IUT * taille 100
              prop CDSALLE * code de la salle *
              prop CAPACITE * nombre de places assises *
              prop CDBAT * code bâtiment *
              prop ETAGE * ras *
              clé CDSALLE
              dclpar CDSALLE
              dclpar CAPACITE
        fcreens
        decpp CAPACITE
              mode 1
              type 3 plage I-NBPERS * capacité d'une salle
                    * de 1 à 500 * ras *
        fdecpp
fin
```

session en traitement par lots sur le projet GESTIONIUT par l'analyste PAOLI

La même session aurait pu se dérouler en conversationnel ; il suffit de remplacer macsibatch par macsi, et de taper chaque ligne sur un terminal.

e) Déroulement d'une session MACSI-P

Tous ces modes d'exploitation sont autorisés quelle que soit l'étape en cours (2 : modélisation, 3 : réalisation ou 4 : utilisation) et leur gestion est décrite dans le dossier technique (Annexe C).

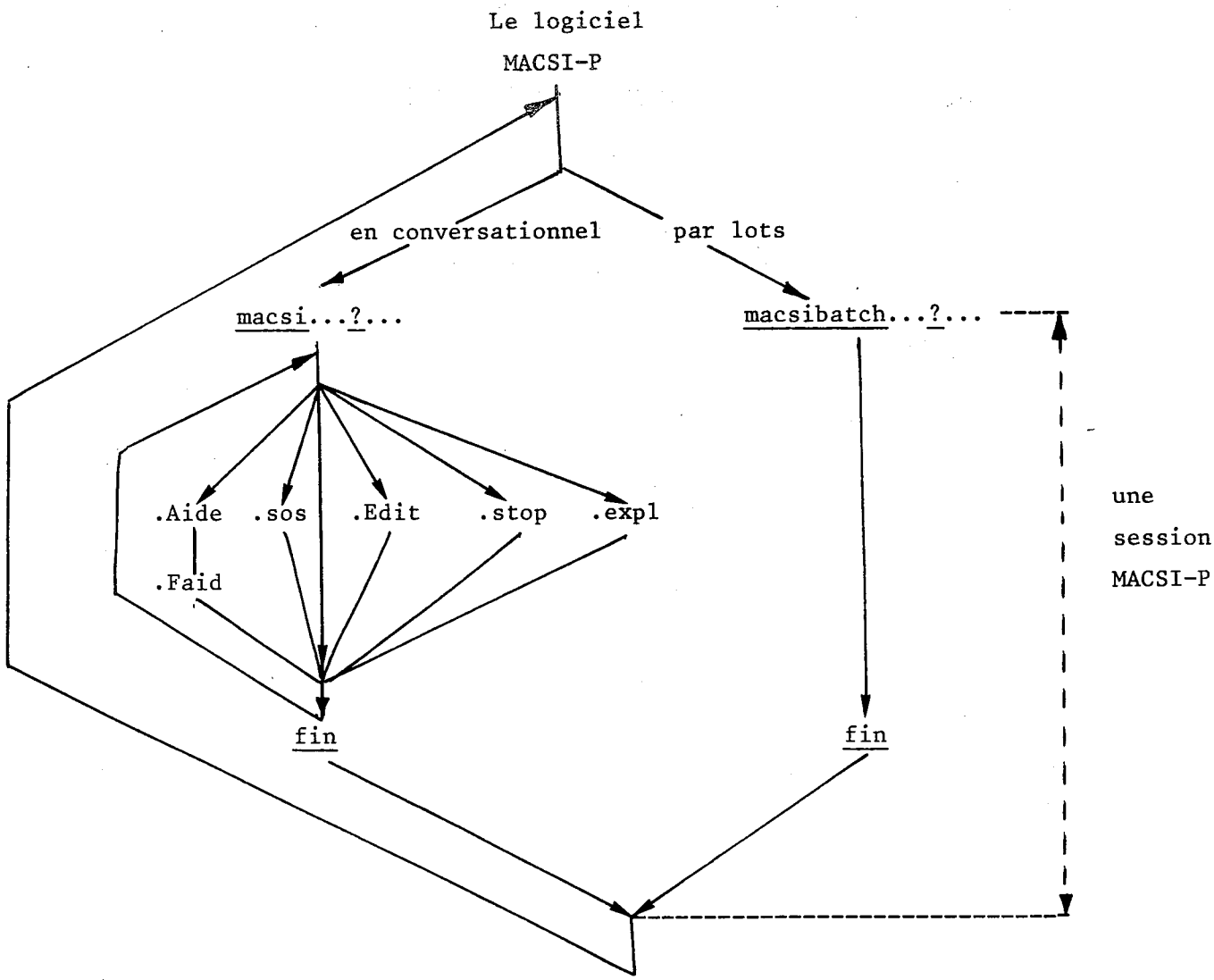


Figure - Déroulement d'une session MACSI-P

IV-2 - ETAPE 1 : INITIALISATION DU LOGICIEL MACSI-P

IV-2-1 - Initialisations de NOYAU et MACSI-P (cf. schéma général § IV-1-1)

Le logiciel MACSI-P se présente comme un ensemble de cartes perforées (ou une bande magnétique) comprenant d'une part une initialisation du processeur de transactions NOYAU, et d'autre part la définition du système MACSI-P.

Elles contiennent essentiellement :

- une définition de la structure Socrate documentaire,
- des descriptions des divers schémas de transactions du langage MACSI-P,
- et un ensemble de programmes utilitaires.

IV-2-2 - Initialisation d'un projet : les instructions CREPRO et EQUIPRO

Pour démarrer un nouveau projet avec le système MACSI-P, il faut, en utilisant les instructions CREPRO et EQUIPRO :

- créer un nouveau projet dans la liste des projets gérés par MACSI-P,
- et lui affecter une liste d'analystes chargés de le décrire.

Le système MACSI-P peut actuellement gérer les spécifications et la documentation de 5 projets simultanément décrits par 100 analystes.

Instruction CREPRO

action : *créer un nouveau projet* dans la liste des projets gérés par MACSI-P et lui *affecter un chef de projet* qui sera l'analyste chargé de constituer l'équipe d'analyse.

syntaxe :

crepro CODEPRO * LIBELLE * chef CODANAL nom * NOM * prénom * PRENOM * fcrepro
(a) (b) (c) (d) (e)

validité : (voir page suivante)

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
CODEPRO	(a)	1	N	- code du nouveau projet, ce code doit être discriminant
LIBELLE	(b)	9	N	- libellé du nouveau projet
CODANAL	(c)	1	N	- code discriminant d'un analyste qui sera le chef du nouveau projet
NOM	(d)	2	N	- nom et prénom du chef de projet CODANAL (c)
PRENOM	(e)	2	N	

messages d'erreurs :

numéro 240 = * vous n'êtes pas l'administrateur de la base MACSI-P *

numéro 241 = * ce code existe déjà *

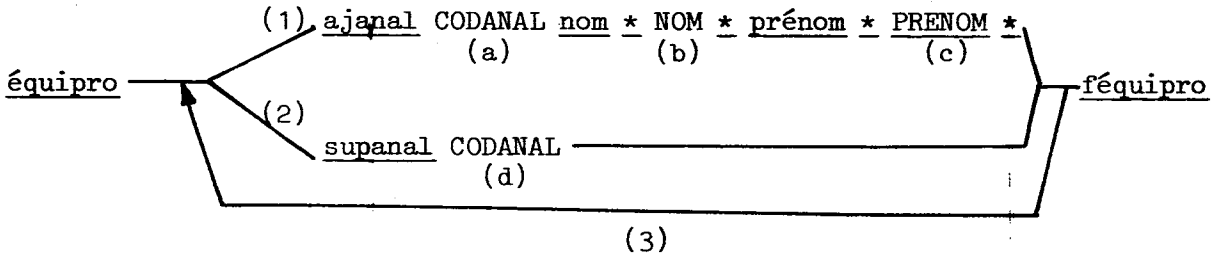
Utilisation :

*crepro GESTIONIUT * gestion de la scolarité à l'IUT2 de Grenoble *
chef PAOLI nom * PAOLI * prénom * CLAUDE *
fcrepro*

Instruction EQUIPRO

action : *constitution de l'équipe* chargée de l'analyse d'un projet, par ajout ou suppression d'un analyste.

syntaxe :



validité : (voir page suivante)

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
/	(1)	/	/	- si le chef du projet veut ajouter un nouvel analyste dans l'équipe
/	(2)	/	/	- si le chef du projet veut supprimer un analyste de l'équipe
CODANAL	(a)	1	N	- CODE, NOM et PRENOM d'un nouvel analyste
NOM	(b)	2	N	
PRENOM	(c)	2	N	
CODANAL	(d)	1	R	- CODE d'un analyste à retrancher de l'équipe
/	(3)	/	/	- si le chef de projet désire faire une autre modification dans son équipe

messages d'erreurs :

numéros 245 = * vous n'êtes pas chef de projet *
246 = * ce code est inconnu *
247 = * ce n'est pas le code d'un analyste *
248 = * le chef analyste ne peut pas être supprimé de l'équipe *

Utilisation :

*équipro ajanal LEF nom * LEFEBVRE * prénom * PATRICK *
ajanal CHABRE nom * CHABRE-PECCOUD * prénom * MONIQUE *
féquipro*

L'instruction CREPRO n'est utilisable que par un analyste particulier : l'administrateur de la base MACSI-P, dont le code est ADMACSIP. Cet analyste doit surveiller chacun des projets et par exemple augmenter les espaces-fichiers lorsqu'ils sont pleins. Mais il doit, lui aussi, utiliser les instructions MACSI ou MACSIBATCH pour accéder à la base documentaire ; le code projet est alors MACSI-P.

L'instruction EQUIPRO n'est utilisable que par le chef du projet qui seul, peut gérer son équipe.

Pour créer le projet, "GESTIONIUT", la séquence sera donc :

```
macsi MACSI-P ADMACSIP ?
      crepro [GESTIONIUT] * gestion de la scolarité à l'IUT2 de
                            Grenoble *
            chef [PAOLI] nom * PAOLI * prénom * CLAUDE
      fcrepo
fin

macsi [GESTIONIUT] [PAOLI] ?
      équipro ajanal LEF nom * Lefebvre * prénom * patrick *
              ajanal CHABRE nom * Chabre-Peccoud * prénom * monique *
      féquipro
fin
```

Le logiciel MACSI-P est ainsi initialisé pour décrire le projet "GESTIONIUT" dont le responsable est "PAOLI" et les analystes sont "LEF" et "CHABRE" ; seules ces trois personnes pourront décrire le projet "GESTIONIUT" jusqu'à une modification éventuelle de l'équipe d'analyse par le chef du projet avec l'instruction EQUIPRO.

IV-3 - ETAPE 2 : ELABORATION DE LA STRUCTURE FONCTIONNELLE D'UNE APPLICATION

IV-3-1 - Introduction (cf. schéma général § IV-1-1)

L'élaboration de la structure fonctionnelle d'une application constitue sa modélisation, c'est-à-dire une description documentée de son modèle fonctionnel des données et des traitements introduits par des réalisations de transactions du langage MACSI-P (cf. chapitre III).

Précisons que nous distinguons trois phases dans la modélisation :

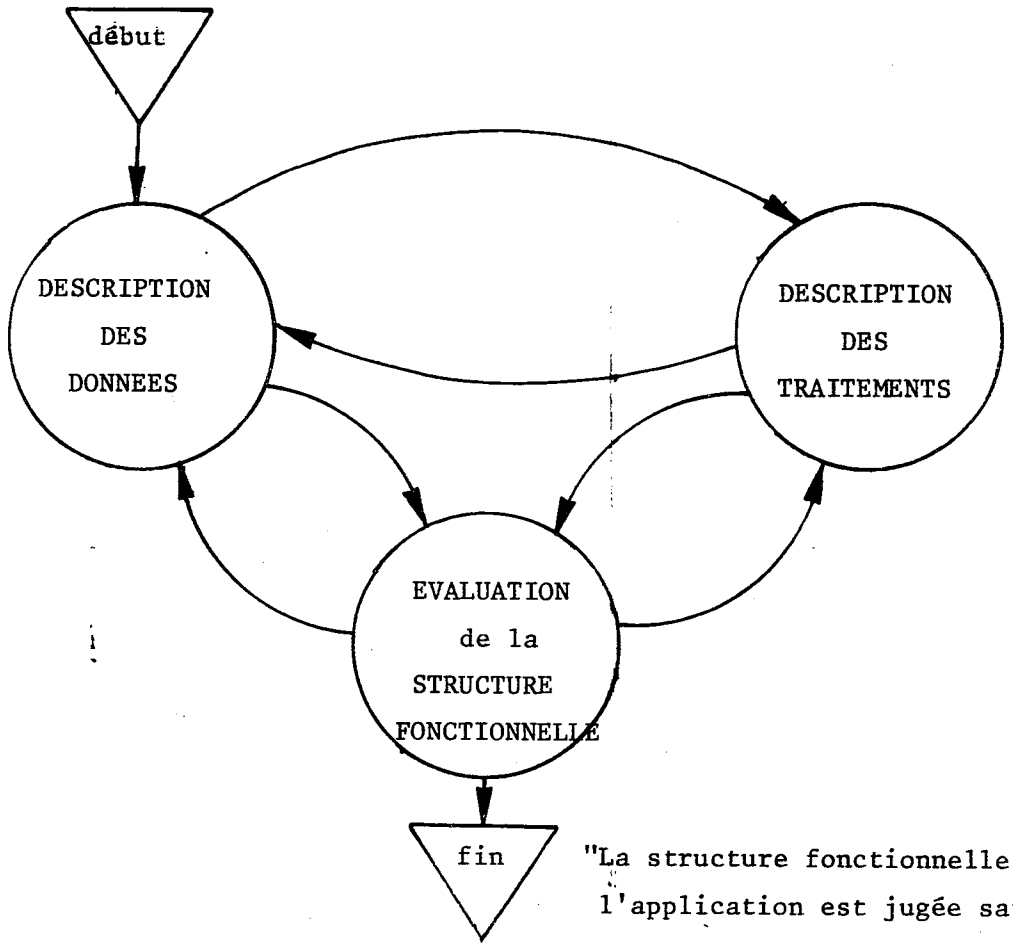


Figure : Descriptions simultanées des DONNEES et des TRAITEMENTS

Rappelons que nous distinguons communément deux classes de méthodes d'approche d'un problème :

- les méthodes d'*approche par les données* qui se développent en deux phases :
  1. définition des données, de leurs relations logiques,
  2. définition des traitements sur ces données.
- les méthodes d'*approche par les traitements* qui consistent à :
  1. définir les traitements,
  2. définir les données nécessaires à l'exécution des traitements.

Nous pensons que ces méthodes sont insuffisantes car d'une part l'obligation d'exécuter dans un ordre strict les définitions ci-dessus est trop contraignante ; et d'autre part l'observation du comportement des analystes dans leurs tâches nous permet de dire qu'il est nécessaire d'autoriser la simultanéité des deux étapes :

- l'analyse des données,
- l'analyse des traitements.

En effet, l'analyste spécifie un projet suivant un processus d'analyse par approximations successives qui concernent tour à tour les données ou les traitements.

Aussi, proposons-nous une méthodologie de modélisation réalisant un compromis entre les deux approches qui permet de décrire simultanément les DONNEES et les TRAITEMENTS :

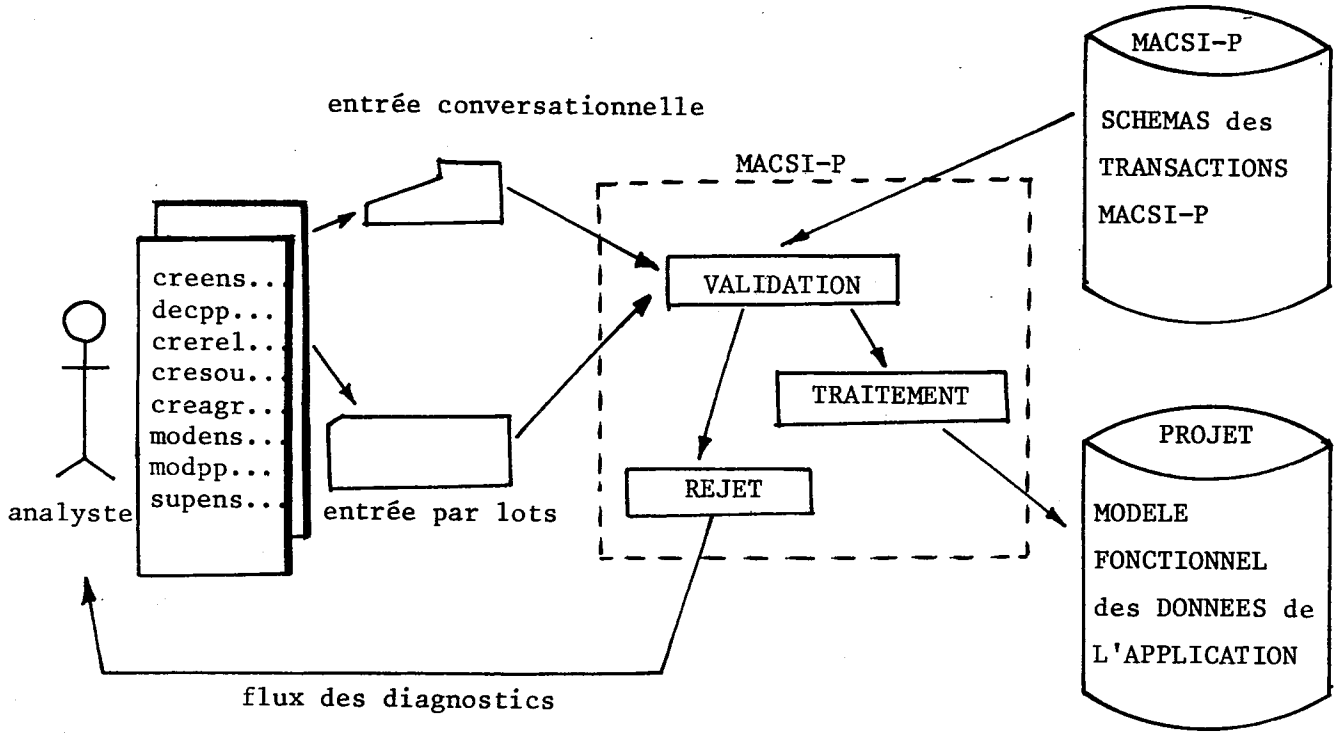
- il est inutile de définir toutes les données et leurs relations logiques avant de procéder à la définition des traitements ;
- au cours de la définition d'un traitement, il est possible de décrire de nouvelles données (nouvelle propriété, nouveau sous-ensemble).

Notons que la modélisation ne peut commencer que si un projet a été créé dans la base MACSI-P, au cours de l'étape 1, pour regrouper toutes les spécifications relatives à l'application.

Nous allons résumer brièvement ces trois phases de modélisation qui ont été décrites dans les chapitres II et III.

IV-3-2 - Description des données

L'écriture de réalisations des transactions CREENS, DECPP, CREREL, CRESOU, CREAGR, MODENS, MODPP et (ou) SUPENS a pour but de décrire et documenter le modèle fonctionnel des données de l'application.



Il est nécessaire de rappeler que les différentes instructions doivent être utilisées dans un certain ordre présenté au § III.2.3.

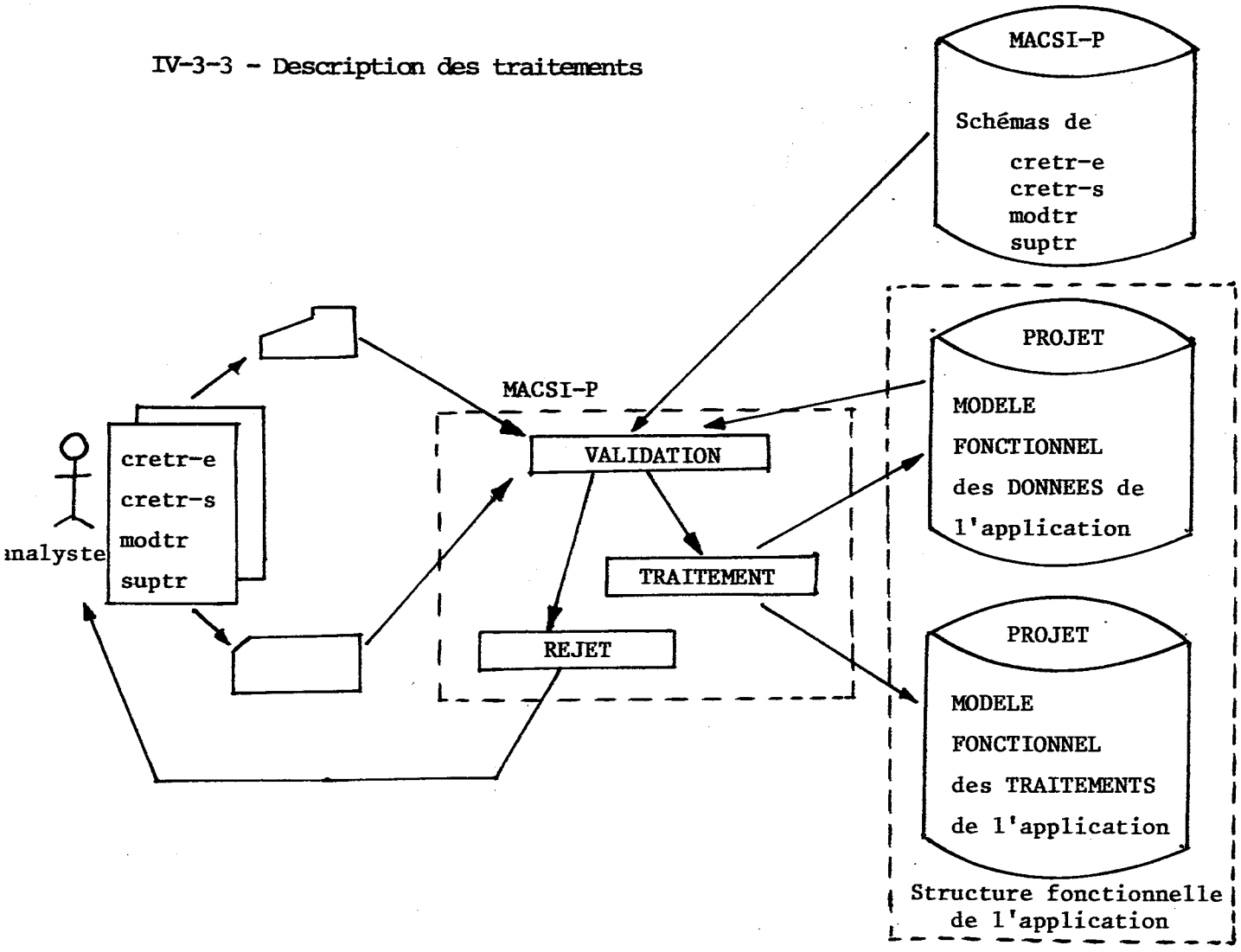
La description des données commence toujours par la création d'ensembles de base avant de pouvoir décrire des propriétés et (ou) créer des relations, des sous-ensembles, des agrégats.

Les instructions proposées assurent une cohérence continue de la description. Mentionnons en particulier que :

- une propriété est toujours rattachée à un et à un seul ensemble,
- une relation est toujours décrite sur des arguments créés,
- le degré de finesse des descriptions est uniforme,
- les modifications ou suppressions respectent cette cohérence.

Des exemples sont donnés dans le chapitre III.

IV-3-3 - Description des traitements

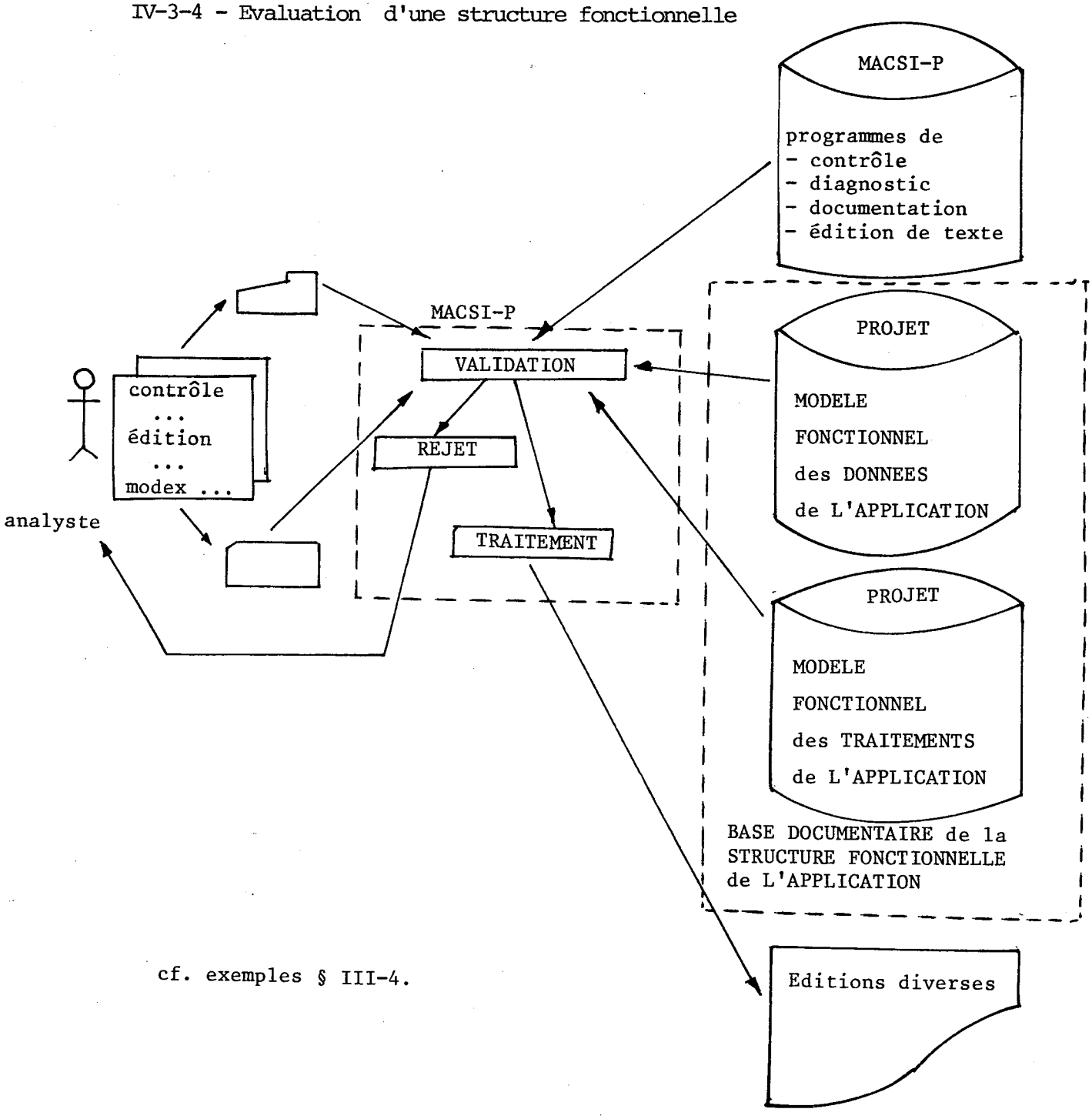


La description des traitements de l'application est réalisée par l'utilisation des transactions MACSI-P : CRETR-E, CRETR-S, MODTR, et (ou) SUPTR (cf. § III-3).

Notons que la structure générale du sous-modèle des données manipulé par une transaction doit être spécifiée préalablement, mais qu'une description d'une transaction peut éventuellement déclarer de nouvelles propriétés dans le modèle des données.

Des exemples sont donnés dans le chapitre III.

IV-3-4 - Evaluation d'une structure fonctionnelle



cf. exemples § III-4.

Après avoir décrit de nombreux ensembles, propriétés ou transactions, il est nécessaire d'effectuer une synthèse de ces descriptions pour en *contrôler* (instruction CONTROLE) l'état d'avancement ou en assurer une *diffusion* (instruction EDITION) auprès des personnes concernées.

Si l'analyste veut apporter des modifications aux textes documentaires de certains constituants, il peut utiliser l'instruction MODEX.

Les principaux objectifs de notre méthode étaient :

- d'obtenir la structure fonctionnelle du futur système d'informations
- d'établir un dossier d'analyse qui soit normalisé.

Les résultats fournis par cette étape sont alors indispensables à la poursuite du projet : il n'est souhaitable de passer à l'étape 3 de réalisation d'un prototype de l'application qu'après un accord précis sur ces résultats entre les différentes personnes concernées (gestionnaires, analystes, réalisateurs, ...).

IV-3-5 - Session de description d'une application

Nous pouvons résumer dans un schéma les divers outils disponibles dans MACSI-P pour décrire une application.

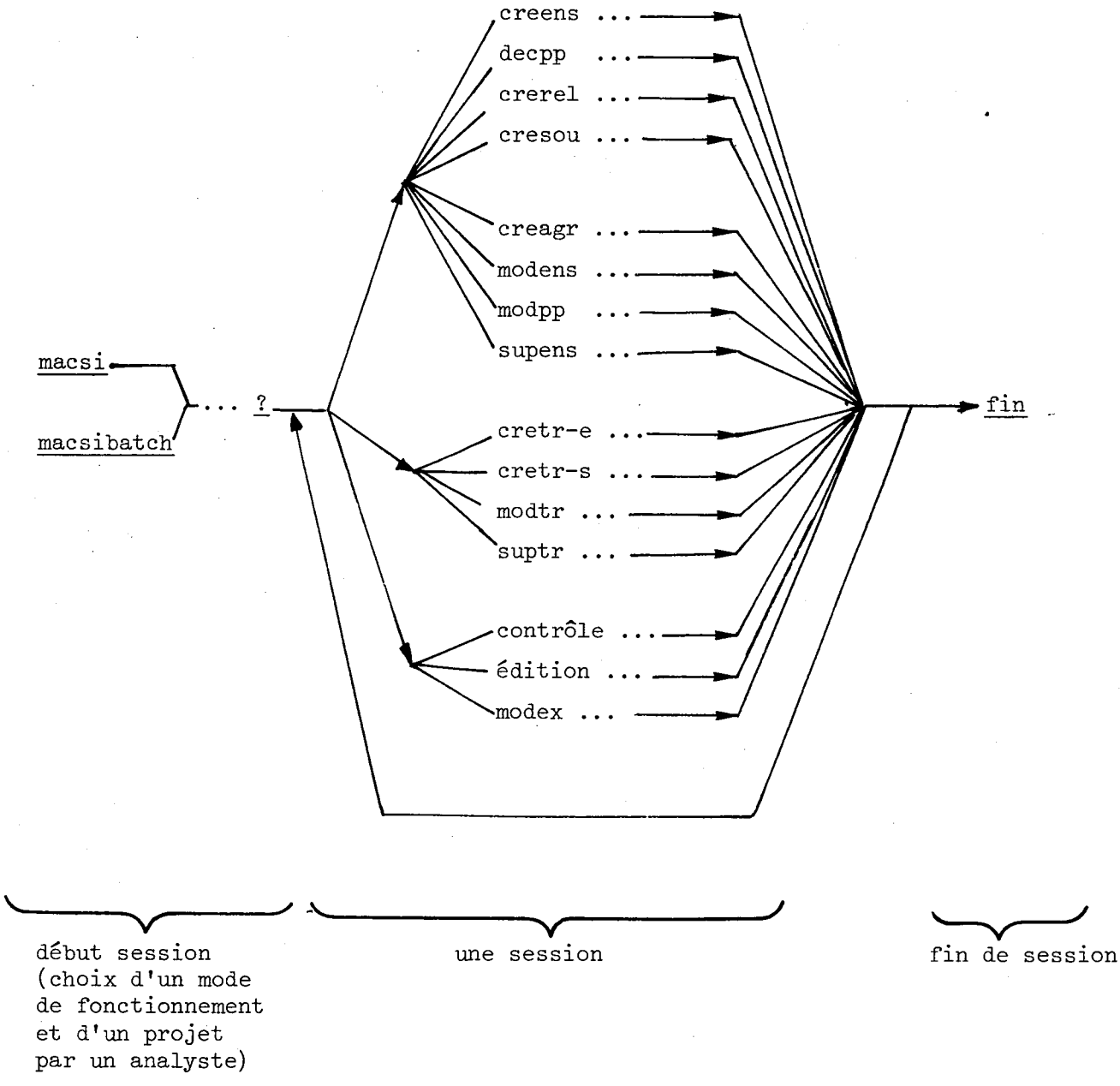


Figure - Description d'une application

IV-4 - ETAPE 3 : REALISATION D'UN PROTOTYPE

IV-4-1 - Choix fondamentaux

L'étape 2 d'utilisation du logiciel MACSI-P correspond à l'analyse d'une application informatique de gestion : elle comprend une description sémantique des données et des traitements et elle s'est achevée par l'impression d'un dossier sur le modèle fonctionnel de l'application. Théoriquement, il y a eu un accord précis entre les utilisateurs et les analystes informaticiens sur les spécifications de l'application contenues dans ce dossier.

Le chef de projet doit alors décider de la façon de poursuivre l'analyse et la réalisation de l'application en choisissant l'une des deux solutions suivantes : (cf. figure § IV-1-1)

Solution 1 : continuer un processus classique de développement du projet en décrivant une ANALYSE ORGANIQUE, la PROGRAMMATION, les TESTS et le LANCEMENT de l'application.

Solution 2 : réaliser un prototype de l'application comme nous le proposons au chapitre I avant de construire l'application réelle.

Dans le cas de la solution 1, le dossier d'analyse, imprimé par MACSI-P en fin d'étape 2, sert de point de départ de la phase d'analyse organique. Le logiciel MACSI-P n'est plus utilisable, sauf pour faire des éditions de dossiers partiels afin de préparer le travail des divers groupes d'analystes-programmeurs (voir instruction EDITION § III-4-2).

Dans ce paragraphe, nous nous intéressons à la solution 2 et nous décrivons comment le logiciel MACSI-P peut être utilisé pour réaliser quasi automatiquement un prototype de l'application spécifiée au cours de l'étape 2.

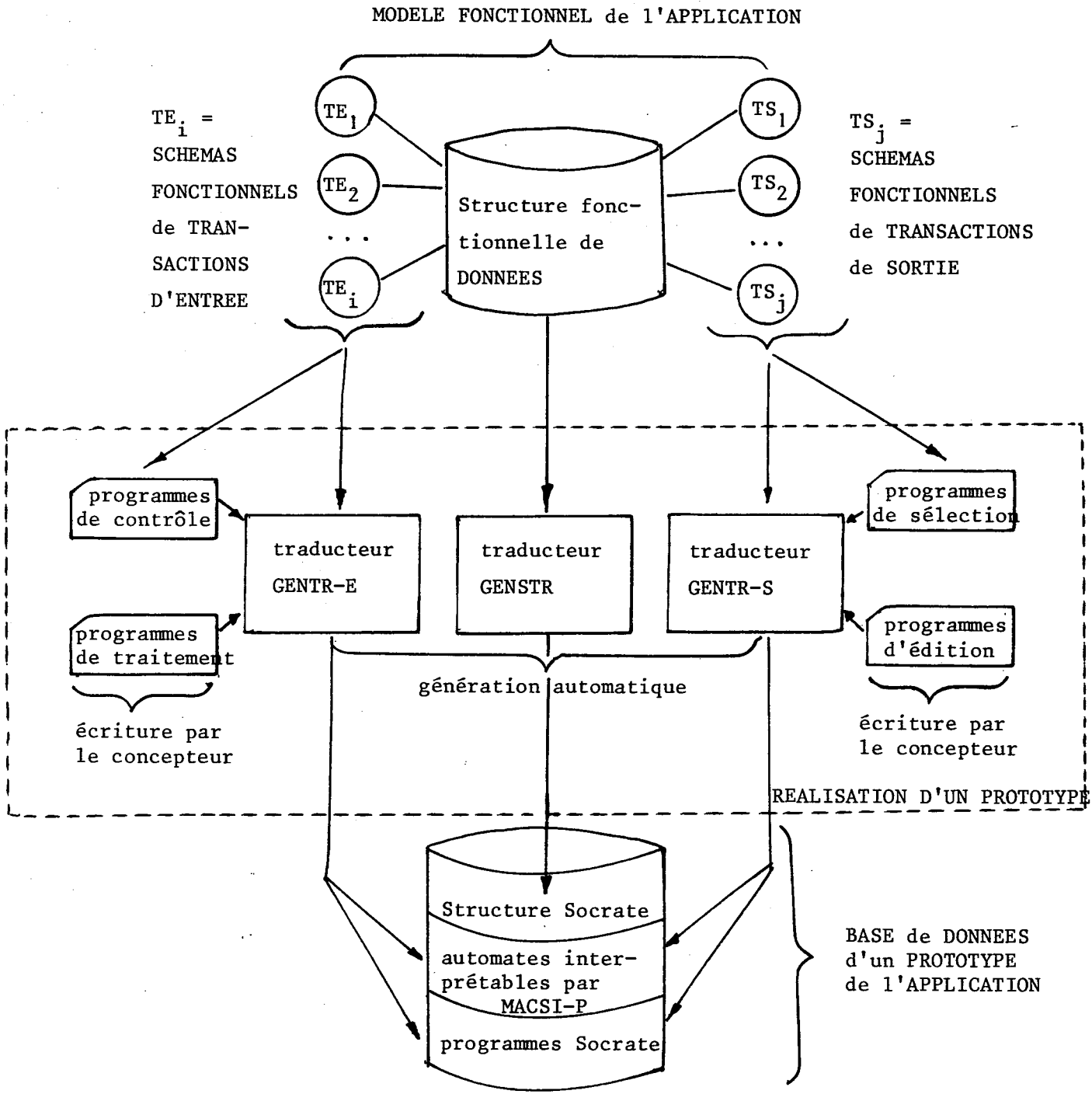
La réalisation d'un prototype est faite en SOCRATE, elle comprend d'une part :

- la génération entièrement automatique d'une structure de données SOCRATE équivalente au modèle fonctionnel des données de l'application spécifié au cours de l'étape précédente,

et d'autre part :

- l'adaptation quasi automatique des transactions (d'entrée et de sortie) décrites dans le modèle fonctionnel des traitements de l'application (étape 2) afin de les rendre exécutables sur la structure Socrate citée ci-dessus.

La figure suivante schématise cette étape de réalisation d'un prototype :



Architecture générale de l'étape 3 de réalisation d'un prototype

La réalisation d'un prototype d'une application se déroule alors logiquement en quatre phases successives :

a) édition du dossier d'analyse de l'application (cf. § IV-3-4) ;

b) génération d'une structure Socrate du prototype par appel du programme GENSTR (cf. § IV-4-2) ;

c) adaptation des transactions d'entrée pour les rendre exécutables sur cette structure Socrate (cf. § IV-4-3) avec :
   - écriture des programmes de CONTROLE spécifiques,
   - écriture du programme de TRAITEMENT de chaque transaction,
   - appel du traducteur GENTR-E ;

d) adaptation des transactions de sortie (cf. § IV-4-4), avec :
   - écriture des programmes de SELECTION,
   - écriture du programme d'EDITION de chaque transaction,
   - appel du traducteur GENTR-S.

Remarque :

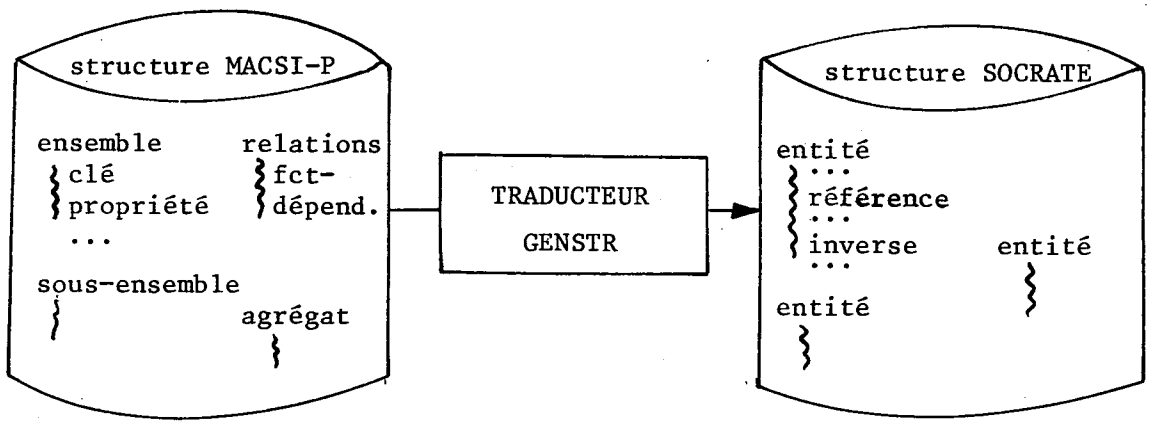
Soulignons que les normes de réalisation que nous proposons pour construire un prototype sont les mêmes que celles que nous avons adoptées pour le logiciel MACSI-P (voir annexe C) et que nous énumérons brièvement :

- mise en oeuvre du système de gestion de base de données SOCRATE ;
- utilisation d'un processeur d'automates d'états finis ("NOYAU") ;
- écriture d'une structure Socrate simple, logique et flexible ;
- gestion centralisée des codes et des textes documentaires (entités DICO et EXPLIC, cf. § C-7) ;
- protection des spécifications et des zones de travail privées pour chaque utilisateur "on-line", permettant un accès simultané de plusieurs analystes sur la base (entité DICINIT, cf. § C-10) ;
- définition de sous-programmes de services généraux pour diminuer la longueur et la complexité d'écriture des programmes.

IV-4-2 - Transformation automatique d'une structure fonctionnelle de données en une structure Socrate* : le programme GENSTR

1 - Hypothèses préalables

Nous voulons proposer aux concepteurs d'un prototype une structure Socrate représentant une implantation logique du modèle fonctionnel des données du projet analysé avec le logiciel MACSI-P.



Cette structure Socrate est la première phase dans la construction d'une "base de données prototype" exploitable à l'aide des transactions décrites par les analystes.

Pour générer une structure Socrate, nous devons choisir le moyen de représenter les différents composants d'un modèle fonctionnel de données :

- les ensembles de base : entités de plus haut niveau ;
- les sous-ensembles : entités, sous-groupes d'une entité ou inverses ;
- les relations binaires ou n-aires : références, inverses ou entité-références ;
- les agrégats : entités ou caractéristiques de travail ;
- les listes de valeurs : explicites ou programmées ;
- ...

Socrate SIRIS 2/SIRIS 3 - CII

Il existe généralement plusieurs structures SOCRATE possibles pour implanter un modèle fonctionnel de données.

Prenons l'exemple suivant où nous avons deux ensembles de base et une relation :

ETUDIANT INSEE, ... ; taille : 100
ENSEIGNEMENT CDENS, ... ; taille : 5
INSCRIPTION (ETUDIANT : F1(0,100), ENSEIGNEMENT : F2(0,1)) ... ;
taille : 100

Proposons quelques solutions de structures Socrate :

structure 1 :

```
entité 100 ETUDIANT
début
    INSEE mot 15 avec clé unique fin
    F2 réfère un ENSEIGNEMENT
fin
```

```
entité 5 ENSEIGNEMENT
début
    CDENS mot 15 avec clé unique fin
    F1 inverse tout ETUDIANT
fin
```

structure 2 :

```
entité 100 ETUDIANT
début
    INSEE ...
    F2 de 1 à 5
fin
```

```
entité 5 ENSEIGNEMENT
début
    CDENS ...
    F1 inverse tout ETUDIANT
fin
```

structure 3 :

```
entité 100 ETUDIANT
début
    INSEE ...
    F2 mot 15
fin
```

```
entité 5 ENSEIGNEMENT
début
    CDENS ...
    F1 inverse tout ETUDIANT
fin
```

structure 4 :

```
entité 100 ETUDIANT
début
    INSEE ...
    F2 réfère un ENSEIGNEMENT
fin
```

```
entité 5 ENSEIGNEMENT
début
    CDENS ...
    entité 100 INSCRIPTION
    début
        F1 réfère un ETUDIANT
    fin
fin
```

structure 5 :

entité 100 ETUDIANT
début
INSEE ...
INSCRIPTIONS réfère une INSCRIPTION
fin

entité 5 ENSEIGNEMENT
début
CDENS ...
INSCRIPTION inverse toute INSCRIPTION
fin

entité 100 INSCRIPTION
début
F1 réfère un ETUDIANT
F2 réfère un ENSEIGNEMENT
fin

D'autres solutions différentes peuvent être déduites en combinant les divers choix d'implantation des relations et de leurs fonctions de dépendance.

Remarquons que pour une fonction de dépendance de cardinal MAX = 1, il existe trois méthodes d'accès possibles :

- par un *pointeur* (réfère) (cf. F2 dans structure 1 et structure 4),
- par identification de l'élément en relation avec la *valeur de son code discriminant* (cf. F2 dans structure 3),
- par identification de l'élément en relation avec le *numéro d'ordre* dans la suite des réalisations de l'entité (cf. F2 dans structure 2).

Si le cardinal MAX d'une fonction de dépendance est supérieur à 1, alors on peut utiliser *une chaîne d'inverses* (cf. F1 dans structures 1, 2 et 3) ou *une entité imbriquée* (cf. F1 dans structure 4) avec les trois possibilités d'accès énoncées ci-dessus.

Mais, si la relation a des propriétés, alors il est plus naturel d'utiliser une solution du type structure 5.

Le générateur envisagé (le programme GENSTR) ne propose qu'une seule solution de structure Socrate compatible avec le modèle fonctionnel de données et qui respecte les règles suivantes :

- elle est la "plus proche" possible du modèle fonctionnel et elle doit traduire la sémantique des données décrites en MACSI-P ;
- elle ne doit pas introduire de nouvelles clauses sémantiques ;
- la documentation de cette structure est nécessaire ;
- la complexité doit être réduite : aucune imbrication d'entités par exemple ;
- aucun chemin d'accès ne doit être privilégié ;
- les ajouts, modifications ou suppressions de caractéristiques ou d'entités doivent être facilités.

La structure 5 relative à l'exemple précédent correspond en partie à ces règles.

Naturellement, nous pouvons donner d'autres critères modifiant le choix d'une structure :

- occupation de place dans l'espace physique,
- longueur des citations interrogatives,
- redondances des chemins d'accès décrits,
- fréquences d'accès aux occurrences d'un ensemble,
- temps de réponse,
- ...

Nous n'avons pas retenu ces derniers critères qui sont souvent incompatibles avec les règles énoncées ci-dessus et qui, à notre idée, ne sont pas primordiaux dans le cas d'une réalisation *d'une base de données prototype.*

De plus, il n'existe pas à l'heure actuelle d'outil permettant, à partir d'une structure fonctionnelle, de déduire la structure de stockage optimale : il faudrait pour cela disposer d'un logiciel de simulation susceptible de prendre en compte simultanément les paramètres ci-dessus ainsi que les contraintes inhérentes au SGBD choisi. Cependant nous proposons un traducteur qui concrétise notre méthodologie d'écriture d'une structure socrate cohérente avec une structure fonctionnelle initiale.

2 - Principes de la transformation

Dans les tableaux suivants, nous exprimons les règles de passage (algorithmes formels) d'une structure MACSI-P à une structure Socrate prototype.

§	DEPART : MACSI-P	OPERATIONS	ARRIVEE : SOCRATE
1		Transformer tout ensemble de base en une entité de niveau 1	
2	CODE, LIBELLE et TAILLE de l'ENSEMBLE de BASE	(voir fin de l'entité § 11)	/* ... LIBELLE ... */ entité TAILLE CODE début
3		Générer une caractéristique de gestion des éléments dans un dictionnaire centralisé	NUMDIC de 1 à 3000
4	CODE d'une propriété clé	Transformer toute clé en une caractéristique simple discriminante	CODE mot 15 avec clé unique fin
5	CODE et TYPE d'une propriété élémentaire, non clé MIN, MAX = valeurs minimales et maximales ~~des bornes ou des listes des~~ valeurs numériques	Une caractéristique simple : TYPE = 1 ou 5 TYPE = 2 ou 6 TYPE = 3, 4, 7 ou 8 TYPE = 9 ou 10 EXPLIC est une entité où sont gérés tous les "textes")	CODE mot 15 CODE mot 30 CODE de MIN à MAX CODE réfère une EXPLIC
6		Générer des caractéristiques simples pour toutes les propriétés particulières à des sous-ensembles de l'ensemble en cours de transformation	
7	CODE d'une propriété clé d'un sous-ensemble	Une caractéristique simple à accès rapide	CODE mot 15 avec clé fin
8	CODE d'une propriété élémentaire, non clé d'un sous-ensemble	Une caractéristique simple suivant le type de la propriété (voir § 5)	CODE mot 15 CODE mot 30 CODE de MIN à MAX CODE réfère une EXPLIC
9		Générer des chaînes d'inverses sur toute relation dont l'ensemble ou un sous-ensemble est argument	
0	CODE d'une relation dont l'ensemble ou un de ses sous-ensembles est argument	Un inverse	CODE inverse tout CODE

§	DEPART : MACSI-P	OPERATIONS	ARRIVEE : SOCRATE
1		Fin de la transformation d'un ensemble	fin
2		Transformer tout sous-ensemble en une chaîne d'inverses sur l'entité qui correspond à l'ensemble "racine"	
3	CODE et LIBELLE d'un sous-ensemble de l'ensemble "racine" CODE2	Un inverse	/* ... LIBELLE ... */ CODE inverse tout CODE2
4		Transformer toute relation en une entité de niveau 1	
5	CODE, LIBELLE et TAILLE d'une relation	idem § 2 et § 3 (fin de l'entité § 23)	/* ... LIBELLE ... */ entité TAILLE CODE début NUMDIC de 1 à 3000
6	CODE d'une propriété clé	idem § 4	CODE mot 15 avec clé unique fin
7		Générer un pointeur (réfère) sur chaque entité équivalente à un argument	
8	CODE d'un argument, FDEP : fonction de dépendance relative à celui-ci	Un pointeur	FDEP réfère un CODARG
9	CODE d'une propriété élémentaire, non clé	idem § 5	CODE mot 15 CODE mot 30 CODE de MIN à MAX CODE réfère une EXPLIC
0	CODE d'une propriété clé d'un sous-ensemble de la relation	idem § 6 et § 7	CODE mot 15 avec clé fin
1	CODE d'une propriété élémentaire, non clé d'un sous-ensemble de la relation	idem § 8	CODE mot 15 CODE mot 30 CODE de MIN à MAX CODE réfère une EXPLIC
2	CODE d'une relation dont la relation ou un de ses sous-ensembles est argument	idem § 9 et § 10	CODE inverse tout CODE

§	DEPART : MACSI-P	OPERATIONS	ARRIVEE : SOCRATE
3		Fin de la transformation d'une relation	fin
4		Transformer tout agrégat en une entité de niveau1	
5	CODE, LIBELLE et TAILLE d'un agrégat	idem § 2 et § 3 (fin de l'entité § 31)	/* ... LIBELLE ... */ entité TAILLE CODE début NUMDIC de 1 à 3000
6	CODE d'une propriété CLE	idem § 4	CODE mot 15 avec clé unique fin
7	CODE d'une propriété élémentaire non CLE	idem § 5	CODE mot 15 CODE mot 30 CODE de MIN à MAX CODE réfère une EXPLIC
3		Générer des chaînes d'inverses pour chaque sous-ensemble agrégé vers l'ensemble "racine"	
9	CODE d'un sous-ensemble agrégé, sous-ensemble d'un ensemble CODE2	Un inverse	CODE inverse tout CODE2
0		idem § 9 et § 10	CODE inverse tout CODE
1	CODE d'une relation dont l'agrégat est un argument	fin de la transformation d'un agrégat	fin

Tableau récapitulatif des algorithmes formels

Structure MACSI-P	Structure SOCRATE
ensemble de base	ENTITE
propriété élémentaire	caractéristique simple
clé	caractéristique discriminante
sous-ensemble	INVERSE vers l'ensemble
relation R entre A, B et C	ENTITE R de références de R vers A de R vers B et de R vers C
agrégat	ENTITE

3 - Remarques méthodologiques et conseils relatifs à l'écriture d'une structure Socrate

Les options prises dans les spécifications et la réalisation du traducteur de structure procèdent de l'impératif de faciliter des modifications ou des extensions d'un prototype construit à partir de la structure générée.

Les règles de traduction que nous venons de décrire dépassent le cadre de la réalisation du traducteur de structure et sont en réalité des principes de base qu'il est souhaitable de respecter en cours d'élaboration d'une base de données que l'on veut suffisamment flexible. Mentionnons en particulier l'écriture des caractéristiques de type "liste de valeurs" et la structuration des ensembles.

Ecriture des listes de valeurs :

Dans Socrate, nous disposons d'une clause sémantique particulière pour définir une caractéristique du type liste de valeurs ; dans ce cas, le système Socrate exécute lui-même une partie de la sémantique des contrôles.

Exemple de structure écrite par un programmeur Socrate :

```
entité 100 ENSEIGNANT
début
     MATIERE (10) (MATH, INFO, ECO, GESTION)
fin
```

Nous préférons écrire (ou générer) une structure Socrate plus simple exprimant une sémantique moins riche et faire exécuter les contrôles de valeurs par des programmes-standards MACSI-P afin que ce système conserve le contrôle du déroulement du programme en cas d'erreur.

Exemple de structure Socrate générée par le traducteur MACSI-P :

```
entité 100 ENSEIGNANT
début
     MATIERE mot 30
fin
```

De plus, MACSI-P gère une réalisation d'une entité LISTE de VALEURS, de code MATIERE et contenant les quatre éléments (MATH, INFO, ECO, GESTION).

```
entité 200 LISTVAL
début
     CODE mot 15
     entité 100 ELEMENT
     début
          VALMOT mot 30
     fin
fin
```

Il suffit de disposer d'un programme de contrôle-standard à deux paramètres, le premier pour le code de la liste (MATIERE par exemple) et le second pour une valeur à contrôler (MATH ou ANGLAIS par exemple) ; appelons CTL-LISTVAL ce programme.

Ce choix de structure de données assure une grande flexibilité de la base Socrate car l'ajout, la suppression ou le changement d'une valeur dans une liste ne nécessite :

- aucune modification de la structure Socrate,
- aucune modification du programme CTL-LISTVAL,
- mais simplement une génération, suppression ou mise à jour d'une réalisation d'un ELEMENT de la liste de valeurs concernée

Structuration des ensembles :

Nous avons choisi de représenter les ensembles de base, les relations et les agrégats comme entités de niveau 1, afin de faciliter des modifications éventuelles de la structure (ajouter de nouveaux ensembles de base, relations, agrégats ou sous-ensembles).Ces modifications sont réalisables par une définition dynamique de structures et ne nécessitent pas une recompilation générale de la structure ni une modification des programmes déjà écrits. Il était donc nécessaire d'éviter les imbrications d'entités.

Transformation des liens logiques en relations d'accès :

De plus, dans le cas d'une relation, le traducteur génère pour chaque argument une fonction d'accès de la relation à l'argument et une fonction d'accès de l'argument vers la relation. Ainsi, aucun argument n'a un accès privilégié, et tous les accès sont autorisés.

Cette technique est indispensable dans une étape de réalisation d'un prototype : il est alors possible d'ajouter ou de modifier des transactions sur le même modèle de données sans remettre en cause la structure Socrate et les chemins d'accès. Ce n'est que dans la construction d'une version finale, avec les enseignements du prototype, qu'il est souhaitable de privilégier ou de supprimer certains chemins d'accès dans un souci d'amélioration des performances ou d'économie d'espace-fichier.

IV-4-3 - Adaptation des transactions d'entrée : le programme GENTR-E

1 - Eléments fondamentaux pour la programmation des transactions d'entrée

L'instruction CRETR-E (cf. § III-3-2) permet la définition d'une nouvelle transaction d'entrée par :

- une documentation générale
- la description de son schéma
- la déclaration et la documentation de programmes de contrôle pour certains champs du schéma ou de programmes qui précédent

(prologues) ou suivent (épilogues) la transaction.

Exemple : On donne une partie du schéma d'une transaction (CRE-ELEV) de création d'un étudiant :

CRE-ELEV :

NUMETUD→...(PS-DISCR)→NOM-PREN→...→RDBLE→...→(PS-RDB)→MOY-PREC→...→ ...

où

PS-DISCR = programme vérifiant que le numéro de l'étudiant à créer est un code discriminant.

PS-RDB = programme indiquant que si et seulement si l'étudiant redouble, il faut alors indiquer la moyenne obtenue l'année précédente.

On peut écrire des réalisations de cette transaction :

*R1 : CRE-ELEV NUMETUD 670754 NOM-PRN*LASCA NOEMIE *RDBLE NON ...*

*R2 : CRE-ELEV NUMEND 690515 NOM-PRN * ZIGFEL JEAN *RDBLE OUI MOY-PREC 8.5 ..*

Dans une "programmation classique", le concepteur du prototype doit écrire, pour chaque transaction, UN PROGRAMME comprenant :

- la saisie d'une réalisation quelconque de cette transaction
- les contrôles standards (plages, listes de valeurs, ...) et/ou particuliers (indirects en général) de chaque champ
- éventuellement, l'impression de messages d'erreurs et/ou de redemande de valeurs (en conversationnel)
- le stockage et la mise à jour définitive des données dans le cas d'une réalisation sans erreurs.

Ce programme est souvent très volumineux et complexe, surtout si le programmeur n'a pas adopté des normes précises de programmation, comme principalement de structurer son programme en séparant les parties relatives aux différentes phases énoncées ci-dessus.

Fonctions assurées par MACSI-P :

Avec le logiciel MACSI-P, ce programme de saisie, contrôle et traitement de toute réalisation d'une transaction ne doit pas être écrit, car MACSI-P assure :

. la saisie d'une réalisation

. l'exécution des contrôles standards sur chaque champ

. le contrôle du bon enchaînement entre les champs

- l'*appel de contrôles spécifiques* pour certains champs
- l'édition d'anomalies éventuelles
- des redemandes de valeurs pour les champs erronés si l'exploitation est conversationnelle
- l'*appel des programmes de traitement* si et seulement si la réalisation est syntaxiquement et sémantiquement correcte.

Programmes à écrire en Socrate par le concepteur du prototype :

Si le système MACSI-P assure de nombreuses fonctions, le programmeur doit cependant écrire POUR CHAQUE TRANSACTION :

- un *PROGRAMME de TRAITEMENT* qui permettra de traiter dans la base de données les réalisations correspondantes et dont l'appel est assuré par le système MACSI-P en fin de validation de celles-ci
- et éventuellement des *PROGRAMMES PROLOGUE* et/ou *EPILOGUE* et/ou de *CONTROLES* particuliers s'ils existent.

Si nous reprenons l'exemple précédent : après avoir défini la transaction CRE-ELEV avec l'instruction CRETR-E, le concepteur du prototype doit simplement écrire les trois petits programmes en Socrate :

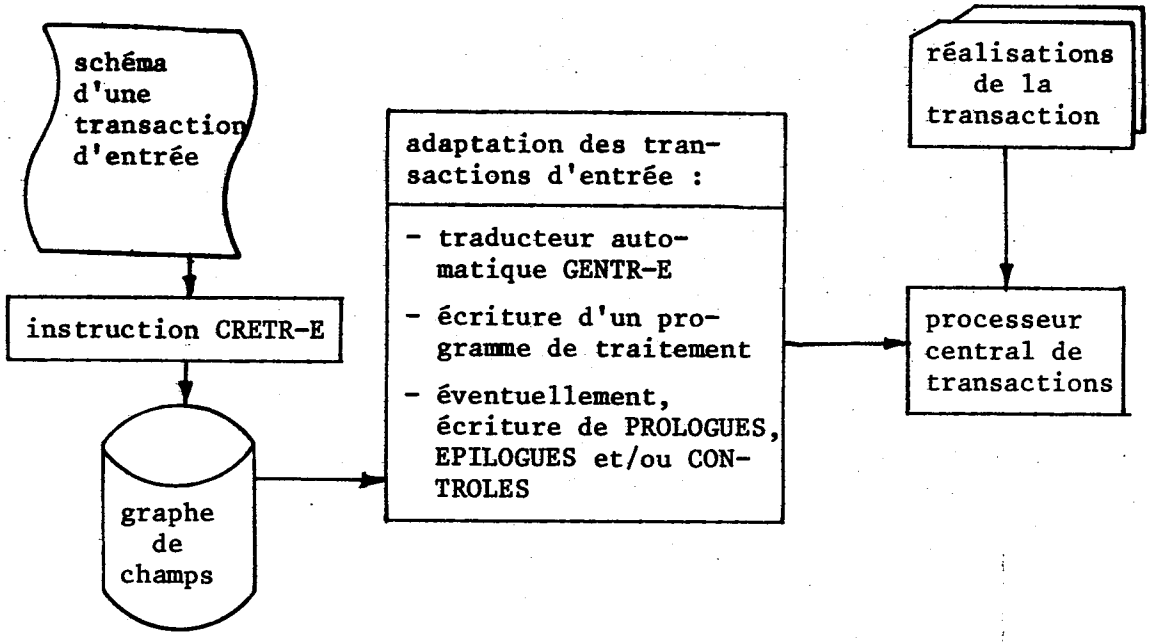
- *le programme de traitement CRE-ELEV*
- *les deux programmes de contrôle PS-DISCR et PS-RDB*

et doit assurer leur insertion dans le système.

Ces programmes écrits par le concepteur sont activés par l'interpréteur MACSI-P, aussi est-il nécessaire de respecter des conventions précises, détaillées en annexe C et résumées ici, quant à leur écriture. Le lecteur peut consulter de nombreux exemples de tels programmes dans l'annexe C.

De plus, il faut transformer chaque schéma de transactions en automate interprétable par le moniteur central ; *cette traduction entièrement automatique est réalisée par le programme GENTR-E.*

Schéma général :



2 - Ecriture des programmes de controle

Ces programmes sont de deux natures distinctes :

- certains sont des programmes de contrôle de la valeur d'un élément d'un schéma
- d'autres sont des programmes de contrôle d'un choix de cheminement dans un schéma.

Les conventions d'écriture de ces programmes sont étudiées au § C-4 ; ils ont la structure suivante :

```
:DEFPRO CONTROLE
:CONTXT XO  D X8 = UN DICO
:EXP
   :
   :
:FDEF ?
```

3 - Ecriture des programmes de traitement

A chaque schéma de transaction est associé un programme de traitement appelé par le moniteur d'enchaînement après chaque réalisation de cette transaction ayant subi avec succès la phase de contrôle.

Le lecteur peut trouver § C-5 toutes les informations et conseils nécessaires à l'écriture de ces programmes.

L'écriture de programmes "PROLOGUES" et "EPILOGUES" doit naturellement respecter les mêmes contraintes et l'appel doit être prévu dans le programme de traitement de la transaction concernée, dont la structure sera :

```
:DEFPRO TRAITEMENT
:CONTXT XO  D X8 = UN DICO
            D X9 = UNE EXPLIC
:EXP
    [exec PROLOGUE]
      :
    [exec EPILOGUE]
:FDEF ?
```

4 - Insertion des programmes de contrôle et des programmes de traitement dans le processeur de transactions

Lorsque le concepteur a écrit ces programmes, il doit alors indiquer au système que ceux-ci réalisent des contrôles ou des traitements mentionnés jusque-là sous certains noms dans des schémas de transaction. La mise en correspondance d'un nom de contrôle ou de traitement et du nom du programme qui l'effectue se fait à l'aide de programmes d'appels associatifs.

. *appel associatif des programmes de contrôle* : (cf. § C-4-6)
Pour insérer un nouveau programme P1 réalisant le contrôle nommé P1, il faut :

1°) ajouter une instruction (voir instruction encadrée) dans le programme U-SEMAN :

```
:DEFPRO U-SEMAN
:EXP
   Faire
        :
        :
      si Z3 = 'P1' alors EXEC P1 sortie fin
        :
        :
   fin
:FDEF ?
```

2°) recompiler ce programme U-SEMAN

. *appel associatif des programmes de traitement* : (cf. § C-5-3)
Dans le cas d'un programme de traitement P2, c'est le programme U-EXECUTION qu'il faut modifier et recompiler :

```
:DEFPRO U-EXECUTION
:EXP
   Faire
        :
        :
      si Z2 = 'P2' alors exec P2 sortie fin
        :
        :
   fin
:FDEF ?
```

Il est recommandé d'utiliser des noms identiques pour dénoter le contrôle (ou le traitement) dans les schémas de transaction et le programme correspondant.

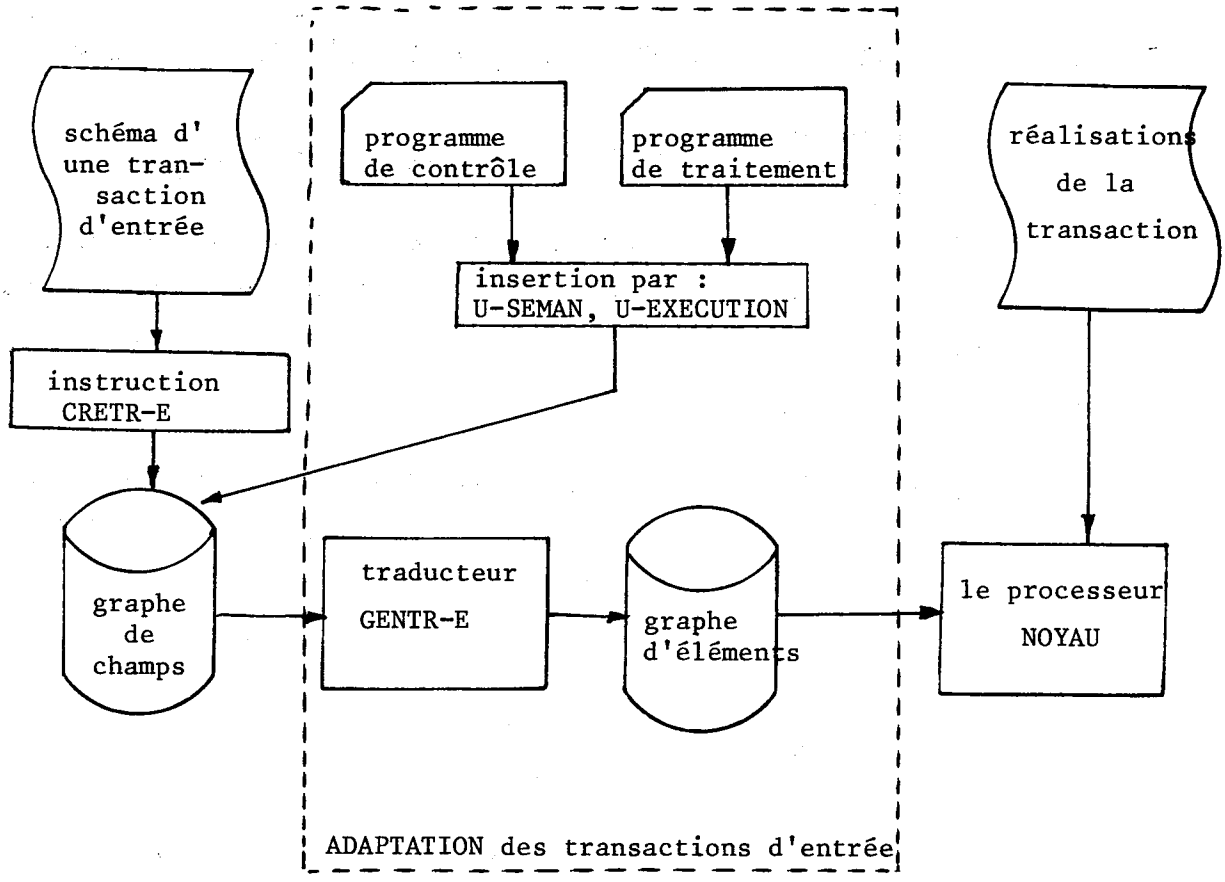
5 - Le traducteur GENTR-E

Le processeur de transactions NOYAU présenté en annexe C ne traite que des réalisations de transactions dont le schéma a été enregistré sous forme de "graphe d'éléments" par la règle MREG (§ C-3).

Or, l'instruction CRETR-E de création des spécifications d'une transaction d'entrée d'une application stocke son schéma sous une forme différente : un "graphe de champs" (§ III-3-1).

Aussi, le programme GENTR-E traduit-il automatiquement un graphe de champs en un graphe d'éléments.

La figure suivante schématise les transformations successives d'un schéma de transaction :



IV-4-4 - Adaptation des transactions de sortie : le programme GENTR-S

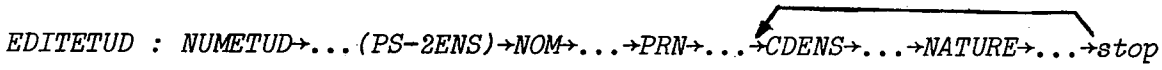
1 - Spécificité des transactions de sortie

Une transaction de sortie est définie par l'instruction CRETR-S (cf. § III-3-3) qui permet essentiellement de décrire son schéma en deux parties :

a) un champ pour *sélectionner* un objet à lister pris dans l'ensemble de référence de la transaction
b) un schéma donnant la *description fonctionnelle du document* à éditer relatif aux objets sélectionnés.

Dans le cas d'une transaction d'édition portant sur *tout* objet de l'ensemble de référence, la partie a) est absente.

Exemple : édition de caractéristiques d'un étudiant qui a suivi les deux années d'enseignement.



champ de lecture : sélection d'un étudiant

écriture : description du document de sortie

où PS-2ENS est un programme qui vérifie que le numéro de l'étudiant (NUMETUD) lu correspond à un étudiant qui a été inscrit dans les deux années.

Remarque :

Au champ de sélection peut être associé un programme particulier de sélection (comme PS-2ENS) qui joue le même rôle qu'un programme de contrôle spécifique dans une transaction d'entrée.

On peut énoncer des demandes de réalisations de la transaction ci-dessus :

R1 : EDITETUD NUMETUD 670754 STOP

R2 : EDITETUD NUMETUD 690236 STOP.

Avec un système de "programmation classique", le concepteur du prototype doit écrire pour chaque transaction de sortie un PROGRAMME qui contient :

- la saisie du champ de sélection d'une réalisation
- les contrôles standards et/ou particuliers de ce champ
- l'impression de messages d'erreurs éventuels
- la redemande de valeurs en conversationnel en cas d'erreurs sur ce champ
- l'édition proprement dite du document demandé.

Le logiciel MACSI-P décharge le concepteur de certaines de ses tâches et facilite son travail pour les autres : il est bien évident que les fonctions assurées par le système proposé sont moins nombreuses dans le cas des transactions de sortie que dans celui des transactions d'entrée (cf. § IV-4-3-1).

Fonctions assurées par MACSI-P :

- Gestion du champ de sélection comme dans une transaction d'entrée, c'est-à-dire saisie et contrôles standards du champ avec appel du programme particulier de sélection éventuellement associé,
- Appel du programme d'édition si et seulement si la phase de sélection s'est bien déroulée.

Programmes à écrire en Socrate par le concepteur du prototype :

Pour chaque transaction de sortie, le programmeur doit écrire :

- un *PROGRAMME D'EDITION* du document décrit fonctionnellement par le schéma,
- et éventuellement des *programmes PROLOGUE et/ou EPILOGUE* et/ou de *SELECTION* particuliers s'ils existent.

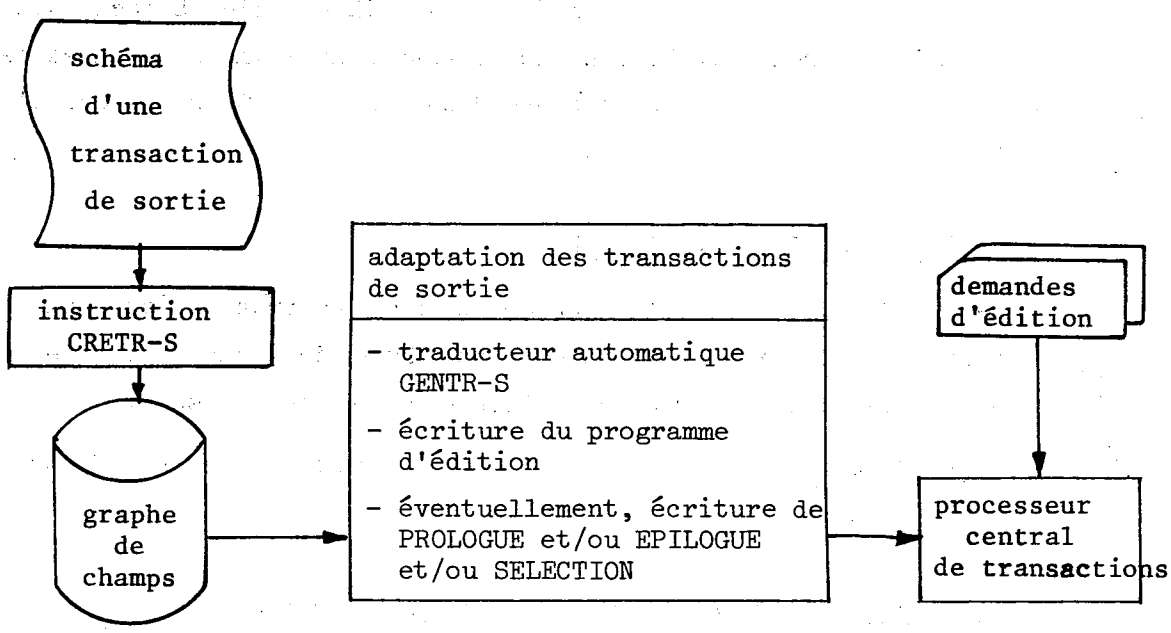
Pour l'exemple précédent, après avoir défini la transaction EDITETUD avec l'instruction CRETR-S, le concepteur doit écrire les deux programmes suivants en Socrate :

- *le programme d'édition EDITETUD pour éditer le document spécifié par la deuxième partie du schéma,*
- *le programme de sélection PS-2ENS.*

L'écriture de ces programmes doit respecter des normes précises quant à l'utilisation des variables et elle est facilitée par l'usage de nombreux programmes de services disponibles dans MACSI-P.

Pour que MACSI-P assure les fonctions énoncées ci-dessus (en particulier la gestion de la lecture et du contrôle du premier champ), il faut transformer chaque schéma de transactions de sortie en un automate qui peut être traité par le moniteur central : *cette transformation est automatiquement réalisée par le programme GENTR-S.*

Schéma général :



2 - Ecriture des programmes de SELECTION

Ces programmes de SELECTION sont de même nature que les programmes de CONTROLE évoqués au § IV-4-3-2, et doivent respecter les contraintes d'écritures données au § C-4 relatives aux programmes de CONTROLE.

Un programme de sélection a alors la structure suivante :

```
:DEFPRO SELECTION
:CONTXT XO   D X8 = un DICO
:EXP
   :
   :
:FDEF ?
```

3 - Ecriture de programmes d'EDITION,de PROLOGUE et d'EPILOGUE

Ces programmes doivent suivre les mêmes règles d'écriture que celles énoncées au § IV-4-3-3 à propos du programme de traitement et du prologue et/ou épilogue d'une transaction d'entrée.

Ces programmes ont la structure suivante :

```
:DEFPRO [ EDITION ]
:CONTXT XO   D X8 = UN DICO
             D X9 = UNE EXPLIC
:EXP
   [exec PROLOGUE]
     :
   [exec EPILOGUE]
:FDEF ?
```

```
:DEFPRO [ PROLOGUE ]  ou EPILOGUE
:CONTXT XO   D X8 = UN DICO
             D X9 = UNE EXPLIC
:EXP
   :
:FDEF ?
```

Pour réaliser les fonctions d'édition propres au logiciel MACSI-P, nous avons utilisé de nombreux outils de service, en particulier pour réaliser l'instruction EDITION (cf. § III-4-2).

Nous proposons que ces programmes de services décrits dans le dossier technique (§ C-9) soient utilisés par le concepteur qui doit écrire les programmes d'édition associés à des transactions de sortie.

Résumons ici les outils disponibles.

a - <u>Une structure élémentaire pour stocker les objets sélectionnés</u>

```
entité 200 INVEDIT
début
     ELEMENT inverse tout DICO
fin
```

Une réalisation de cette entité est une chaîne d'inverses ELEMENT qui permet de "pointer" un ou plusieurs objets de dictionnaire centralisé. Il est conseillé de n'enregistrer dans une même chaîne d'inverses que des objets appartenant à un même ensemble (objets qui ont une même nature).

b - des programmes ou des macro-instructions dont nous donnons brièvement le nom et la fonction principale dans le tableau suivant :

nom de l'outil	rôle de l'outil
ECRIT	imprimer une ligne, gérer la pagination, remplacer l'instruction Socrate ECRIRE
CADRE	éditer un titre encadré
EDLICO	éditer le CODE et le LIBELLE d'un objet
EDTXT	éditer les TEXTES explicatifs d'un objet
EDINVERSE	éditer les CODES et LIBELLES de tous les objets liés par une chaîne d'INVERSES à l'objet sélectionné courant
XEDS	pour activer un même programme d'édition sur tous les objets sélectionnés dans UNE chaîne d'inverses de l'entité INVEDIT

c - ainsi que de nombreux outils pour faciliter la mise en page d'un document (EN-TETE, LTIRE, BL, BP, HP, ...).

4 - Insertion des programmes de sélection et d'édition dans le processeur de transactions

Comme pour les programmes de CONTROLE et de TRAITEMENT des transactions d'entrée (cf. § IV-4-3-4), le concepteur doit indiquer au système que ces programmes écrits réalisent des sélections ou éditions nommées dans des schémas de transaction.

La correspondance entre un nom cité dans des schémas et un nom (il est recommandé qu'il soit le même) d'un programme Socrate est faite dans un programme d'appel associatif modifié par le concepteur et recompilé.

. L'insertion d'un programme SI de sélection qui réalise la sélection dénommée SI dans un schéma est faite par le programme U-SEMAN (cf. § C-4-6) ainsi modifié :

```
:DEFPRO U-SEMAN
:EXP
    faire
         :
         :
         si Z3 = 'SI' alors EXEC SI sortie fin
         :
         :
    fin
:FDEF ?
```

. Dans le cas d'un programme d'édition (EDIT), c'est au programme U-EXECUTION qu'il faut ajouter une instruction (cf. § C-5-3) :

```
:DEFPRO U-EXECUTION
:EXP
    faire
         :
         :
          si Z2 = 'EDIT' alors EXEC EDIT sortie fin
         :
         :
    fin
:FEDF ?
```

5 - Le traducteur GENTR-S

Ce programme réalise la traduction automatique d'un schéma d'une transaction de sortie, enregistré sous forme d'un "graphe de champs" en un graphe d'éléments (cf. § C-6-4) qui peut être traité par le processeur de transactions NOYAU.

On peut schématiser sur la figure suivante les phases d'adaptation d'une transaction de sortie :

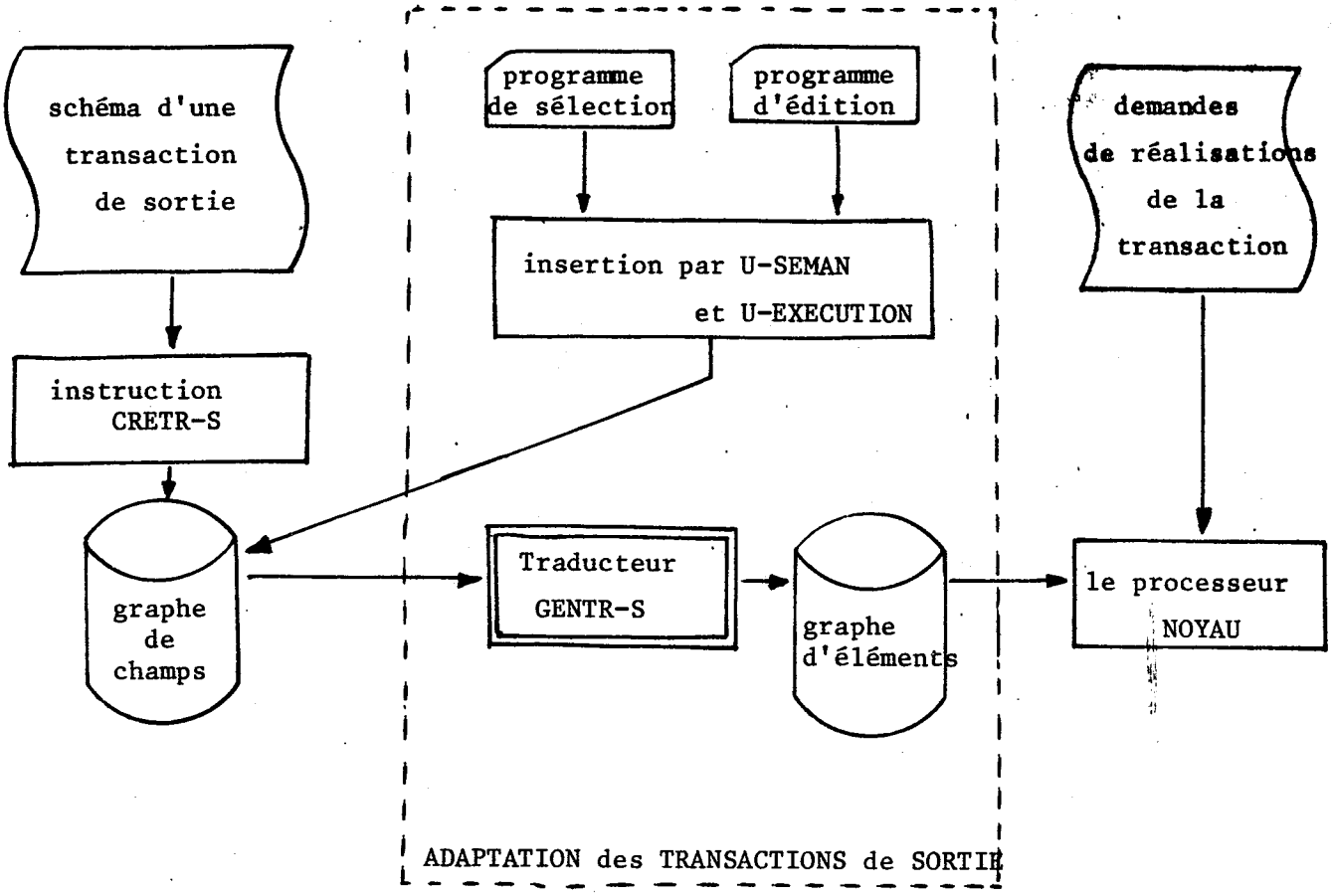
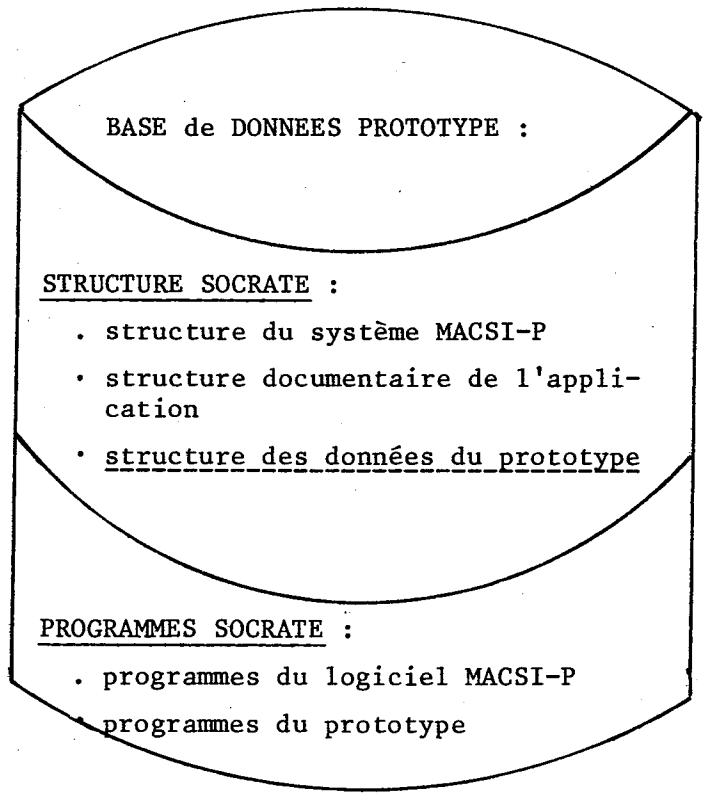


schéma d'une transaction de sortie
programme de sélection
programme d'édition
demandes de réalisations de la transaction
insertion par U-SEMAN et U-EXECUTION
instruction CRETR-S
graphe de champs
Traducteur GENTR-S
graphe d'éléments
le processeur NOYAU
ADAPTATION des TRANSACTIONS de SORTIE

IV-5 - ETAPE 4 : UTILISATION D'UN PROTOTYPE

IV-5-1 - Un prototype = une base de données Socrate

Un prototype construit avec le logiciel MACSI-P (étape 3, § IV-4) est une BASE DE DONNEES SOCRATE :



Cette base de données prototype contient :

- la structure de base et les programmes nécessaires au fonctionnement du logiciel MACSI-P et du processeur de transactions (des schémas de transactions, des programmes de parcours de graphes, ...)
- la structure documentaire de l'application issue de la phase d'élaboration d'une structure fonctionnelle (des ENSEMBLES, PROPRIETES, SCHEMAS de TRANSACTIONS, ...)
- la structure de données générée automatiquement par le traducteur GENSTR ainsi que les programmes associés aux transactions de l'application (des CONTROLES, TRAITEMENTS, SELECTIONS, EDITIONS, ...).

Tous ces éléments structuraux ou programmes se situent à des niveaux différents d'élaboration et d'utilisation dans cette base de données, car la construction s'est effectuée par étapes successives autour du SGBD Socrate.

IV-5-2 - Déclaration des utilisateurs d'un prototype : instruction EQUITEST

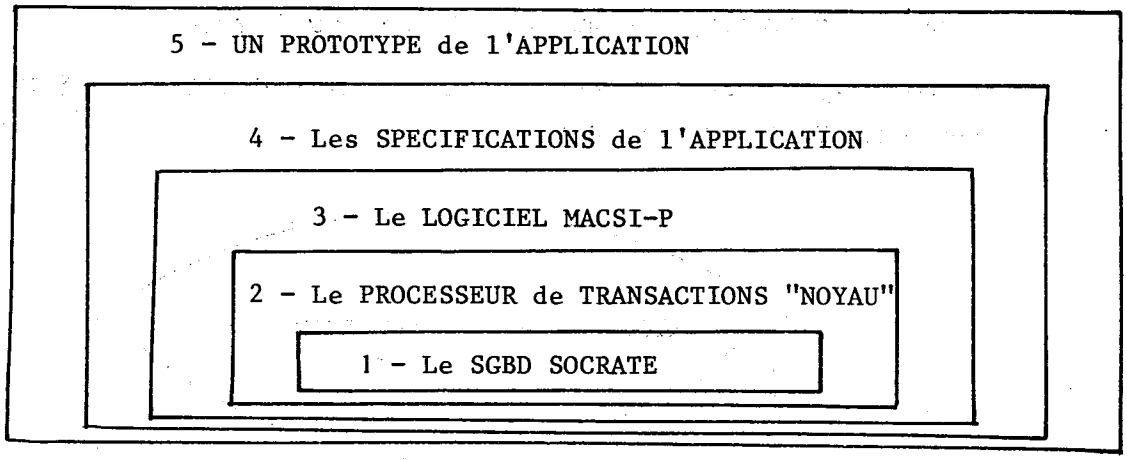
Les personnes qui utilisent le prototype doivent être représentatives des véritables utilisateurs de la version finale de l'application, c'est-à-dire des personnes concernées dans leur activité quotidienne par son fonctionnement. Ces personnes devront juger de l'adaptation de l'outil à leurs besoins et à leurs contraintes de travail lorsqu'ils ont à s'en servir effectivement.

Au chapitre I, nous avons justifié la construction d'un prototype et sa mise en service pour les futurs utilisateurs de l'application, en indiquant les principales utilisations :

- familiariser les utilisateurs de la future application et assurer leur formation pour les tâches dont ils seront responsables,
- tester les transactions spécifiées et valider les schémas logiques des traitements,
- mesurer la bonne compréhension des tâches demandées et vérifier que l'acquisition des données nécessaires à la constitution d'une réalisation d'une transaction d'entrée est possible,
- enregistrer les remarques et les demandes de modifications exprimées par les utilisateurs,
- ...

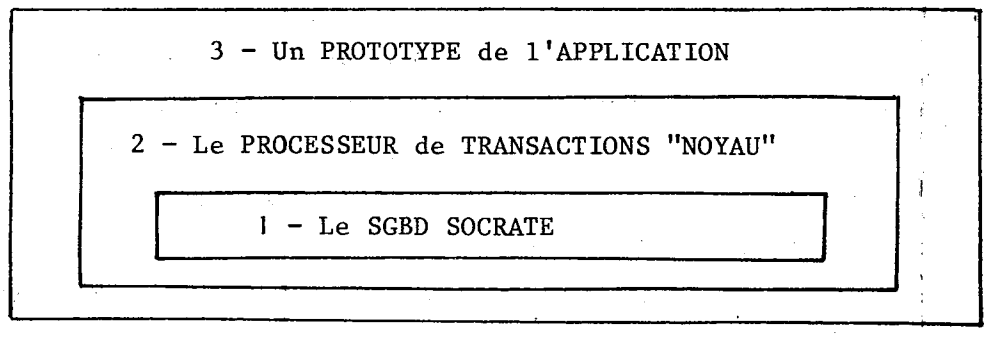
Aussi, une liste de ces utilisateurs clairement établie est-elle un préalable à toute utilisation d'un prototype ; c'est le rôle de l'instruction EQUITEST.

Résumons ces divers niveaux dans la figure suivante :



Un des objectifs poursuivis est que l'utilisation d'un prototype soit indépendante de tous ces niveaux : le prototype doit être utilisable par les futurs utilisateurs de l'application réelle dans les mêmes conditions que le logiciel MACSI-P est disponible pour les analystes du projet ; ceci dans un double souci de simplicité et d'efficacité.

Ce but est atteint dans la mesure où dans l'étape de réalisation du prototype, § IV-4, les activations des trois traducteurs automatiques, GENSTR (§ IV-4-2-2), GENTR-E (§ IV-4-3-5) et GENTR-S (§ IV-4-4-5), conduisent à obtenir un prototype directement exploitable par le processeur central de transactions. On se trouve alors devant une solution moins hiérarchisée résumée par la nouvelle figure :



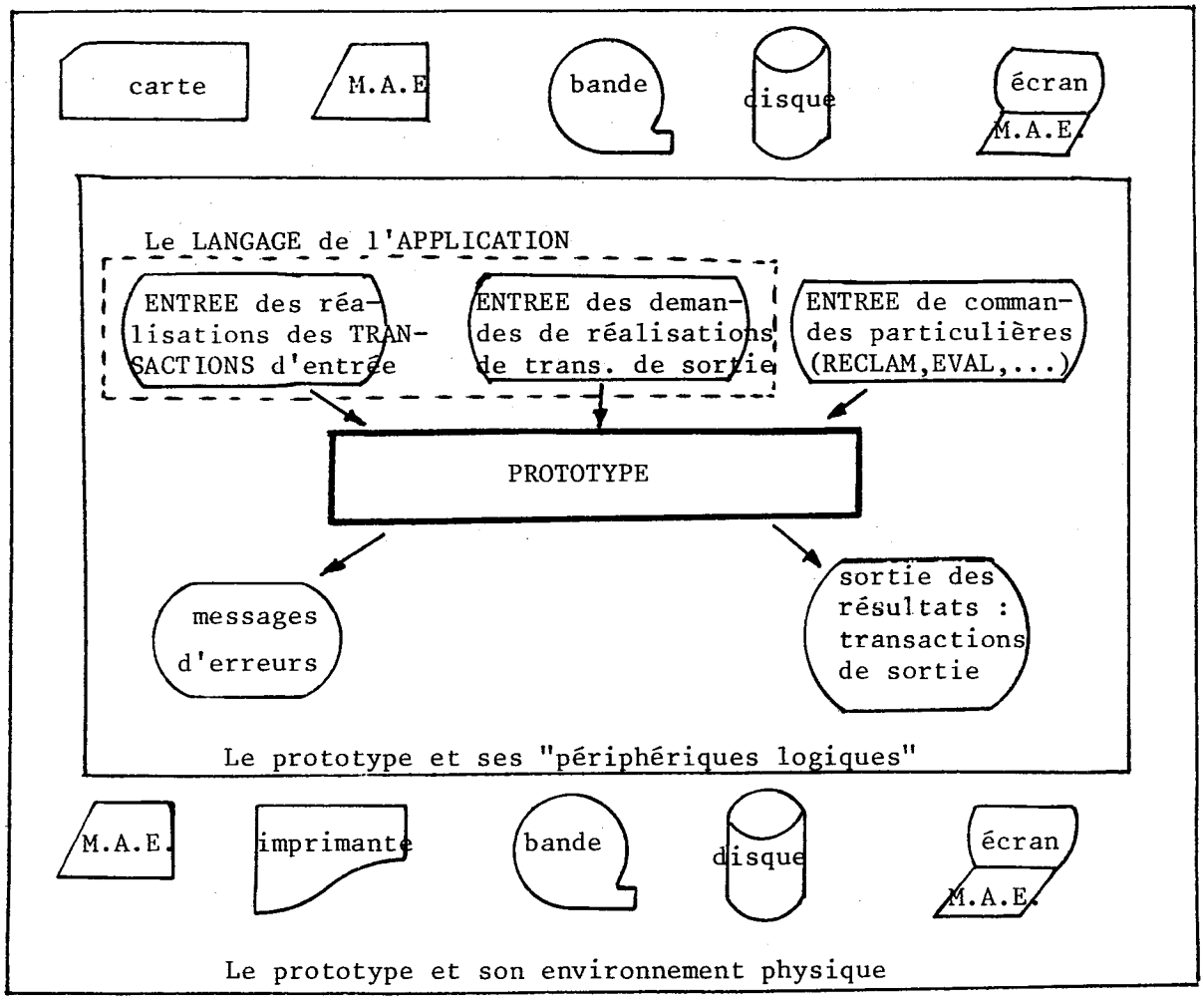
Comme pour tout produit informatique, il faut constituer *son dossier d'utilisation* qui doit préciser :

- qui va utiliser le prototype ?
- comment doit-on l'utiliser ?
- et peut-on évaluer son utilisation ?

Dans les paragraphes suivants, nous indiquons brièvement comment il est possible de décrire un tel dossier d'utilisation d'un prototype et nous montrons qu'il est pratiquement identique à celui du logiciel MACSI-P.

Or, l'utilisation du logiciel MACSI-P prouve que les analystes n'ont aucune contrainte informatique provenant des deux niveaux inférieurs (① SGBD Socrate + ② processeur de transactions) : ils doivent simplement respecter l'écriture de réalisations de transactions conformes aux schémas des transactions de MACSI-P (§ IV-3).

Aussi, donnons-nous des moyens d'exploiter le prototype, afin que les utilisateurs le voient sous un aspect plus réel et plus simple, en le considérant comme un outil informatique dont le *langage de manipulation est entièrement défini par les transactions de l'application.*

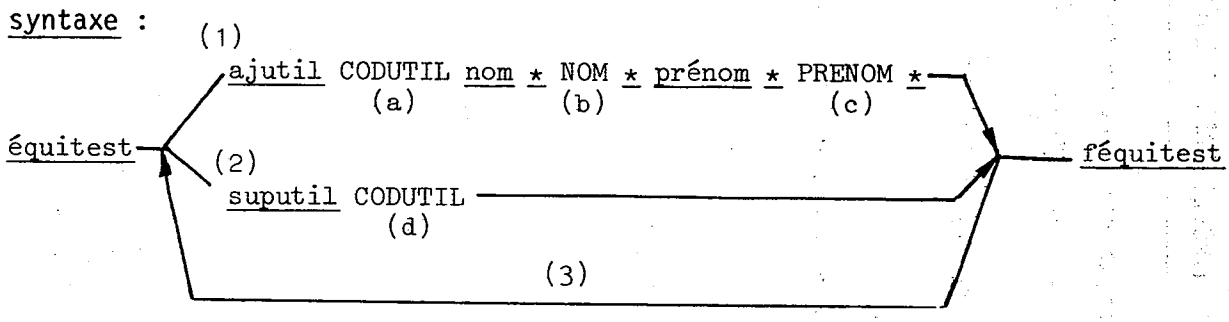


- Le prototype et les utilisateurs -

Instruction EQUITEST

action : *constitution de l'équipe* chargée de l'utilisation d'un prototype, par ajout ou suppression d'un utilisateur.

syntaxe :



validité : (voir page suivante)

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
/	(1)	/	/	- lorsque le chef du projet veut ajouter un nouvel utilisateur dans l'équipe
/	(2)	/	/	- lorsque le chef du projet veut supprimer un utilisateur de l'équipe
CODUTIL	(a)	1	N	- CODE, NOM et PRENOM d'un nouvel utilisateur du prototype
NOM	(b)	2	N	
PRENOM	(c)	2	N	
CODUTIL	(d)	1	R	- CODE d'un utilisateur à retrancher de l'équipe
/	(3)	/	/	- si le chef de projet désire faire une autre modification dans son équipe

utilisations :

L'instruction EQUITEST ne peut être utilisée que par le chef du projet (défini par une instruction crepro ..., cf. § IV-2) qui, lui seul, peut constituer l'équipe des utilisateurs qui testeront le prototype.

Pour décrire les utilisateurs du prototype de l'application "GESTIONIUT", après avoir indiqué au système le code du projet et le code du chef du projet, on écrirait, par exemple, la séquence suivante :

```
macsibatch GESTIONIUT PAOLI ?
    équitest ajutil UTFRB nom * FRAIMBAULT * prénom * MADELEINE *
             ajutil UTDPS nom * DESPESSE * prénom * CHRISTIANE *
             ajutil UTEDR nom * DE ROSA * prénom * EVELYNE *
             ajutil UTPAN nom * PANEL * prénom * DANIEL *
    féquitest
fin
```

L'instruction EQUITEST est pour le prototype l'équivalent de l'instruction EQUIPRO pour le logiciel MACSI-P (§ IV-2-2).

IV-5-3 - Une session d'utilisation d'un prototype : les instructions PROTO et PROTOBATCH

Après avoir indiqué qui doit utiliser le prototype construit (instruction EQUITEST), on peut décrire comment on l'utilise.

Un prototype construit avec le logiciel MACSI-P peut fonctionner sous deux modes :

- en conversationnel,
- en traitement par lots.

Le choix d'un de ces modes est réalisé à l'aide des deux instructions PROTO et PROTOBATCH présentées dans ce paragraphes.

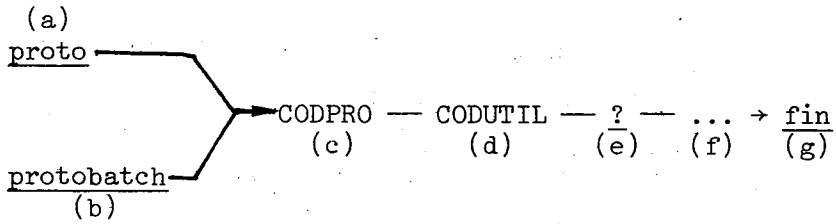
Au préalable, nous pouvons affirmer que les remarques faites au sujet du fonctionnement de MACSI-P (cf. § IV-1-4) s'appliquent aussi à celui d'un prototype :

- les transactions sont utilisables sous les deux modes
- l'écriture de réalisations de transactions se fait en format libre : l'analyse syntaxique et sémantique est dirigée par les mots clefs de la transaction
- le mode conversationnel autorise une correction immédiate des erreurs
- une modification de la base de données n'est réalisée que si la réalisation de la transaction a été jugée sans erreurs syntaxiques ou sémantiques
- des fonctions d'assistance peuvent être appelées en conversationnel afin de connaître à tout moment ce que l'on doit ou peut faire (.SOS, .AIDE, .FAID, .EXPL, .EDIT, .SOPT).

Instructions PROTO et PROTOBATCH

action : *démarrer une session* de travail sur un prototype d'une application en mode conversationnel ou en traitement par lots.Une session concerne un "prototype d'un projet" pour lequel un "utilisateur" fournit des réalisations de transactions.

syntaxe :



validité : (voir page suivante)

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
proto	(a)	0	/	- début d'une session d'utilisation d'un prototype en conversationnel
protobatch	(b)	0	/	- début d'une session d'utilisation d'un prototype en batch - une session en batch est constituée d'un paquet de cartes ou d'une saisie sur bande magnétique. Le premier enregistrement doit commencer par protobatch, le dernier ne contiendra que FIN. Les enregistrements doivent faire 80 caractères
CODPRO	(c)	1	R	- code d'un projet déjà déclaré dont on veut utiliser le prototype
CODUTIL	(d)	1	R	- code d'un utilisateur déclaré et affecté au prototype du projet CODPRO (c)
?	(e)	0	/	- marque la fin de l'accès à un projet et le début de la session proprement dite
...	(f)	/	/	- liste d'instructions du prototype
fin	(g)	0	/	- fin d'une session d'utilisation d'un prototype

Utilisations :

Soit une session batch de déclaration des utilisateurs du prototype du projet "GESTION IUT".

```
                                    projet    chef du projet
                  macsibatch GESTIONIUT PAOLI ?

une session   ⎧  équitest ajutil UTFRB nom * FRAIMBAULT * prénom * MADELEINE *
"MACSI"       ⎪           ajutil UTDPS nom * DESPESSE * prénom * CHRISTIANE *
              ⎩           ajutil UTEDR nom * DE ROSA * prénom * EVELYNE *
                  féquitest
                  fin
```

Soit une autre session batch d'utilisation de transaction de ce prototype :

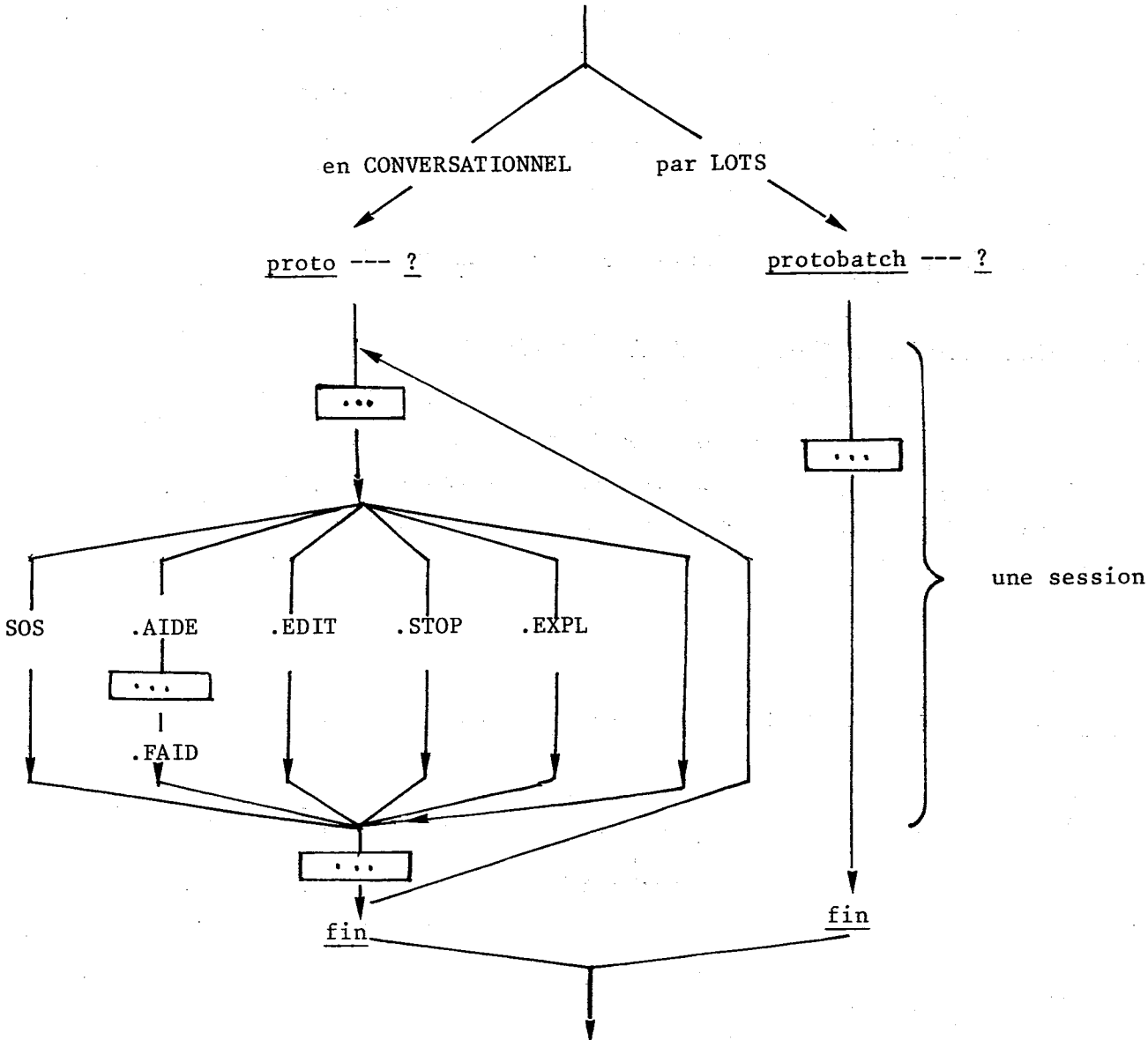
```
                                    projet    utilisateur
                  protobatch GESTIONIUT UTDPS ?

                  cre-élev numetud 670754                  ⎫  réalisation
                           nom-prn * LASCA-NOEMIE *        ⎬  d'une transac-
                           rdble NON                       ⎪  tion d'entrée
                           ...                             ⎭
                  fcre-élev
une session       cre-élev numetud 690515                  ⎫
"prototype"               nom-prn * ZIGFEL JEAN *         ⎬  réalisation
                          rdble OUI                       ⎪  d'une transac-
                          moy-prec 8,5                    ⎪  tion d'entrée
                          ...                             ⎭
                  fcre-élev
                  edit-élev numetud 670754 stop           ⎫  demande de réali-
                  ...                                     ⎭  sation d'une tran-
                                                             saction de sortie
                  fin
```

ou une session en conversationnel :

une session "prototype"

```
proto GESTIONIUT UTDPS ?
    cre-élev numetud [.SOS]
    --- VALEUR NUMETUD * numéro d'inscription d'un étudiant *
    - 670754
      ...
fin
```

Schéma de déroulement d'une session d'utilisation d'un prototype



... : des réalisations de transactions du prototype d'applications

Dans le cadre de l'écriture d'un dossier d'utilisation d'un prototype, il faudrait alors présenter toutes les transactions permises en entrée ou en sortie du prototype comme nous l'avons fait au chapitre III pour MACSI-P.

IV-5-4 - Evaluation de l'utilisation d'un prototype : les instructions RECLAM et EVAL

L'utilisation d'un prototype est considérée comme la simulation d'une application et elle doit permettre d'une part :

- d'évaluer le degré de satisfaction des utilisateurs afin de faire des modifications éventuellement souhaitées,

et d'autre part :

- d'enregistrer et d'éditer des informations statistiques pour préparer la réalisation de la version finale de cette application.

Ce sont les deux objectifs essentiels retenus dans MACSI-P (cf. chapitre I).

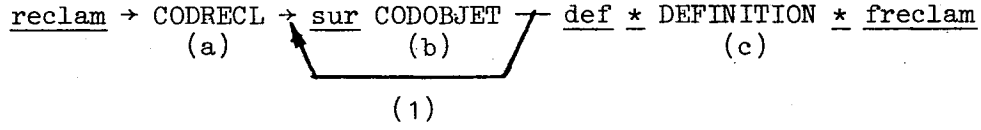
Etude du degré de satisfaction des utilisateurs :

Nous avons jugé nécessaire l'introduction d'un outil (l'instruction RECLAM) pour gérer automatiquement les textes exprimant toutes les difficultés rencontrées par les utilisateurs d'un prototype d'un projet donné. L'analyse de ces textes pour évaluer ce degré de satisfaction sort du cadre de cette étude.

Instruction RECLAM

action : *faire une réclamation* concernant des "objets" d'un projet.

syntaxe :



validité : (voir page suivante)

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
CODRECL	(a)	1	N	- code discriminant de la réclamation
CODOBJET	(b)	1	R	- code d'un "objet" du projet sur lequel porte la réclamation - l'objet d'une réclamation peut être n'importe quel constituant du modèle fonctionnel de l'application : . une propriété dont la valeur est souvent difficile à obtenir . une propriété dont le type est mal choisi . une transaction de sortie qui ne restitue pas le document souhaité . une transaction d'entrée qui demande des informations dispersées dans des postes de travail distincts et éloignés . une transaction d'entrée d'informations redondantes . ...
/	(1)	/	/	- si la réclamation porte sur plusieurs objets
DEFINITION	(c)	10	N	- texte définissant et justifiant une réclamation - exemples : des questions, des remarques, des souhaits, des nouvelles propositions, ...

utilisations :

Cette instruction est utilisée au cours d'une session normale d'accès à un prototype d'un projet par un utilisateur.

projet utilisateur

proto GESTIONIUT UTEDR ?
reclam REC17 sur CRE-ELEV
sur MOY-PREC } *objets de la réclamation*
*def * lorsqu'un étudiant est redoublant, il ne connaît que rarement la moyenne de ses notes de l'année précédente : quelle réponse doit-on donner ? un blanc, U, RAS, ... **
freclam
fin

où CRE-ELEV est une transaction d'entrée (cf. § IV-4-3-1)
et MOY-PREC est un champ (une propriété dont la valeur est à saisir) de CRE-ELEV.

Préparation d'une version finale :

La *description* d'un prototype et l'*évaluation de son utilisation* sont des éléments fondamentaux pour réaliser une version finale d'une application dans les meilleures conditions possibles. L'instruction EVAL assure l'appel de programmes d'édition des évaluations qui permettent de faire le point sur l'état de la base de données du prototype ou sur la façon dont celui-ci a été utilisé .

L'utilisation de l'instruction EVAL est à combiner avec l'instruction d'EDITION des spécifications (cf. § III-4-2).

Nous ne pouvons pas donner ici une liste complète de tous les programmes d'évaluation, car elle n'est pas limitée. En effet, on peut toujours imaginer de nouvelles évaluations utiles avant d'engager la programmation d'une version finale. De même, il faudrait pouvoir définir des algorithmes de passage d'un prototype à une version finale en prenant en compte les contraintes inhérentes au système de programmation choisi pour cette dernière version.

Néanmoins, nous proposons quelques programmes dont l'emploi nous semble nécessaire quel que soit le projet étudié.
Ce sont :

EV-TRNON : liste des transactions non utilisées
intérêt : ces transactions ont-elles une utilité ?

EV-TR-HST : liste des transactions utilisées classées par leur nombre respectif d'utilisation
intérêt : guider le choix d'une structure de données adaptée qui privilégie les chemins d'accès décrits par les transactions les plus utilisées et guider le choix du langage de programmation de celles-ci.

EV-TRUT1 : liste des transactions utilisées en indiquant :
- le nombre d'utilisations
- les codes des différents utilisateurs
- les dates d'utilisation

intérêt : contrôle de l'accès au prototype.

EV-TRUT2 : liste identique à EV-TRUT1 classée par date

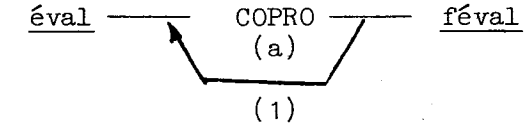
EV-TRUT3 : liste identique à EV-TRUT1

EV-UTNON : liste des utilisateurs affectés au prototype et qui n'ont utilisé aucune transaction
intérêt : doit-on les supprimer ? doit-on décrire de nouvelles transactions ? ...

EV-OBRCL : liste des objets concernés par une ou plusieurs réclamations
intérêt : préparer les modifications éventuelles du prototype.

Instructions EVAL

action : demander l'exécution d'un ou plusieurs programmes d'évaluation de l'utilisation d'un prototype.

syntaxe : 

validité : (voir page suivante)

Zone	Repère	Type	R/N	COMMENTAIRES ET CONTROLES PARTICULIERS
CODPRO	(a)	5	R	- code d'un programme d'évaluation - il doit appartenir à la liste des programmes d'évaluation : EV-TRNON, EV-TRNST, EV-TRUT1, EV-TRUT2, EV-TRUT3, EV-UTNON, EV-OBRCL, ...
/	(1)	/	/	- si l'on désire faire exécuter plusieurs programmes d'évaluation.

Utilisation :

Après de nombreuses sessions d'utilisations du prototype dans l'environnement de l'application future, il est souhaitable d'utiliser l'instruction EVAL suivante :

projet *chef du projet*

```
macsibatch GESTIONIUT PAOLI ?
      éval EV-OBRCL
           EV-UTNON
           EV-TRNON
           EV-TRHST
           EV-TRUT1
      féval
fin
```

} *liste des programmes d'évaluation à exécuter*

CHAPITRE V

CONCLUSION

V-1 - Réflexions sur la technique du prototype

V-2 - Perspectives d'utilisation et de développement du modèle

Nos objectifs étaient de proposer un MODELE, un LANGAGE et des OUTILS pour :

- représenter un système d'informations comme une collection d'ensembles sur lesquels sont définies des transactions,
- utiliser cette représentation pour fabriquer un prototype de ce système d'informations.

Au terme de la description du projet MACSI-P, peut-être pouvons-nous proposer au lecteur quelques réflexions sur le bilan provisoire que nous sommes amenés à faire concernant principalement le modèle de systèmes d'informations et la technique du prototype.

V-1 - REFLEXIONS SUR LA TECHNIQUE DU PROTOTYPE

Au début de ce travail, nous avons développé l'idée de réaliser, avant la construction d'un système informatique, un modèle, fonctionnellement équivalent, pour permettre une meilleure analyse en soumettant des solutions programmées aux utilisateurs, afin d'éviter les remises en cause trop tardives. L'expérience nous conforte dans l'intérêt de cet objectif.

Nous avons proposé des outils pour construire (§IV-4) et utiliser (§IV-5) un prototype. Mais nous devons faire deux constats qui nous suggèrent des prolongements au travail présenté ici.

a) Il devient nécessaire de réfléchir à une procédure complète de validation d'un prototype en déterminant essentiellement les conditions de son utilisation, les critères d'évaluation et les mesures possibles.

Si nous avons esquissé ce problème de normes de validation au §IV-5, l'expérience nous permet de mieux cerner les améliorations souhaitables.

- les conditions d'utilisation doivent définir comment le prototype est mis à la disposition des utilisateurs pour sa validation ; il faudrait définir les normes pour évaluer leurs réactions. Déjà, nous avons rencontré certains problèmes :

  . la souplesse d'utilisation du prototype peut amener les utilisateurs à exiger une souplesse équivalente pour l'application réelle souvent incompatible avec des volumes de données beaucoup plus importants ou des choix techniques d'exploitation imposés (cf. [L14] , [L15] et [A12]).

  . la technique de programmation employée pour réaliser un prototype peut conduire à fausser son évaluation. En effet, avec les outils proposés (annexe C), les problèmes de conflits d'accès ne peuvent être simulés ; ce qui nous conduit à écrire certains programmes non partageables pour le prototype alors qu'ils devraient en réalité représenter des fonctions multi-utilisateurs dans la version opérationnelle.

- Trois critères d'évaluation doivent être précisés en priorité :

  . la satisfaction obtenue dont l'appréciation est tout à fait subjective. Avec l'instruction RECLAM (§IV-5-4), nous pensons qu'il est plus facile d'enregistrer l'insatisfaction des utilisateurs qui se prononcent toujours sur ce qui ne marche pas et rarement sur ce qui va très bien.

. le délai d'exécution d'une fonction doit être du même ordre de grandeur dans le prototype et dans la version finale. Si cette condition n'est pas réalisée, il faut pouvoir évaluer quel est le délai maximum acceptable.

. le coût de réalisation du prototype permet-il d'estimer celui de la version finale ? Nous ne connaissons aucune méthode de calcul donnant une réponse à cette question qui suppose d'abord que la notion de coût de réalisation soit clairement posée. Mais nous pouvons espérer que le prototype permette d'évaluer à quel prix maximum un utilisateur d'une fonction du prototype accepterait de financer sa réalisation.

- Les mesures possibles sont spécifiques à chaque prototype ; nous les avons évoquées au § I.3.4. Mais il faut noter que l'élaboration et l'utilisation de ces mesures sont très délicates et leurs résultats peuvent être fortement liés aux organes technologiques utilisés pour le prototype ; par exemple, des mesures effectuées sur un poste de travail associé à un prototype sont certainement sans intérêt pour la version opérationnelle si le poste finalement retenu est totalement différent.

  Aussi, serait-il souhaitable d'envisager le développement d'une forme de périphérique extrêmement modulaire pour un prototype, permettant d'avoir au niveau des matériels d'entrée-sortie le même degré de flexibilité que pour le logiciel (possibilités de touches de fonctions en nombre variable, affichage sur écran ou sur papier, ...).

b) Il reste à étudier le passage d'un prototype à une version opérationnelle à l'image du passage d'un produit industriel mis au point en laboratoire vers la chaîne de fabrication. Il y a encore beaucoup à faire dans ce domaine. Mais deux questions fondamentales se posent déjà :

- comment concevoir des processus assistés par ordinateur permettant d'acheminer un prototype vers un produit opérationnel ?
- le fait de construire un prototype avec un SGBD influence-t-il fortement le choix du logiciel utilisé pour le produit final et comment ?

C'est un art confié plus spécialement à un ingénieur-système que de répondre à ces questions. Cependant, nous pouvons donner quelques éléments de réponse :

. ces processus assistés, s'ils existent, doivent aider l'informaticien à choisir une structure physique des données, à prévoir les performances du système réel, à programmer les fonctions retenues ;

. si la construction d'un prototype est possible avec des logiciels très souples, des SGBD de haut niveau, elle n'implique pas forcément un constat de faisabilité de la version finale avec des SGBD plus rustiques et des outils logiciels moins évolués ;

. enfin, les prototypes auxquels nous avons participé [L14][L15] correspondent tous à une première introduction de l'informatique dans une organisation. Il faudrait aussi étudier des cas concrets où des versions successives d'une application opérationnelle doivent servir à concevoir et réaliser une nouvelle version tenant compte des enseignements tirés des versions précédentes.

V-2 - PERSPECTIVES D'UTILISATION ET DE DEVELOPPEMENT DU MODELE

Notre proposition de modèle se situe dans le cadre actuel des recherches concernant la conception des systèmes d'informations. Elle n'est qu'une tentative parmi d'autres [L1][L3][L9][L24][L25] [A7][A8][A9][A11][A15][A23][A26] ; elle ne vise donc pas à l'universalité.

Ces nombreux travaux nous confirment dans l'idée qu'il n'y a pas une méthode unique pour concevoir des systèmes informatiques même si par ailleurs un effort de normalisation des concepts employés doit être entrepris : laissons ce rôle à des associations savantes (AFCET, IFIP, CODASYL, ANSI, AFNOR, ...).

Cette normalisation n'existant pas, nous avons dû définir longuement tous les concepts introduits dans notre modèle (cf. chapitre II) bien que de nombreux termes employés ont des équivalents dans d'autres travaux mentionnés.

La méthode associée à un modèle et à un langage demande une mise au point non seulement lors de sa conception mais aussi pour réaliser les supports de son enseignement ; elle n'est pertinente que si elle peut être transmise. Nous avons l'expérience de l'enseignement des concepts pourtant simples et rigoureux de la programmation structurée [L6][L7] [L11][L19] qui rencontre cependant des difficultés : les définitions précises des notions de "programme", "sous-programme" et "interface" sont-elles suffisantes pour réaliser un bon découpage du problème ?

Le projet MACSI-P autorise quelques espoirs concernant l'enseignement de l'analyse. Le modèle proposé dans le chapitre II a fait l'objet d'un cours d'analyse en D.U.T. d'informatique ; cet enseignement a mis en évidence :

- la possibilité de présenter un modèle de système d'informations indépendamment des langages et des outils de mise en oeuvre ;
- l'intérêt de cette approche avant tout enseignement relatif aux systèmes de gestion de bases de données ;
- l'importance d'un formalisme mathématique rigoureux pour définir les concepts ;
- la nécessité de certaines améliorations du modèle.

Il est par exemple indispensable d'assurer la liaison avec le modèle MACSI-1 (cf. annexe B, [L5][L22]), pour permettre une analyse fonctionnelle et organique d'un projet sans discontinuité : après une

approche modulaire d'un projet avec les concepts MACSI-1, il est alors souhaitable d'utiliser MACSI-P pour préciser l'analyse des données et des traitements de l'application concernée et éventuellement construire un prototype.

Cette liaison entre les deux systèmes MACSI-1 et MACSI-P, bien que prévue au niveau de la conception, n'est encore pas réalisée techniquement, mais nous l'envisageons à brève échéance.

Enfin certaines extensions du modèle seront certainement nécessaires à l'occasion de confrontations avec des problèmes particuliers à résoudre. Nous avons montré dans le chapitre II, avec une présentation de notre modèle par enrichissements successifs, la démarche à suivre pour introduire une extension :

- définir sans ambiguité les nouveaux concepts qu'elle propose,
- préciser l'utilité de l'extension dans la représentation d'un système d'informations,
- s'assurer de la compatibilité de l'extension avec les concepts précédemment utilisés.

ANNEXE A

PRÉSENTATION DE QUELQUES MODÈLES DE SYSTÈMES D'INFORMATIONS

ANNEXE A1

LE MODÈLE RELATIONNEL DE E.F. CODD

CODD propose [A9] de représenter logiquement une Base de Données par une collection de tableaux (de relations n-aires) variable dans le temps pour améliorer l'indépendance des programmes d'application vis-à-vis de modifications dans la représentation des données. Dans ce but, il développe un formalisme relationnel et une algèbre relationnelle.

Nous présentons les principales définitions données par C. DELOBEL dans [L9][L10] et reprises dans [L18].

A1-1 - Relations n-aires : définitions

- Champs et constituants (DOMAINS & ATTRIBUTES)

Un champ est un ensemble U non vide. Pour exprimer que l'élément a appartient à U, on note :

$$a \in U$$

Un constituant (noté par une lettre majuscule) est une variable qui peut momentanément prendre une valeur. Attribuer un champ U à un constituant X, c'est décider de choisir les affectations de X parmi les valeurs de U ; on note cette attribution :

$$X :\in U$$

Affecter une valeur a à un constituant X est noté :

$$X := a$$

- Constituants composés : p-uplet de constituants

Un constituant composé est un p-uplet de constituants, c'est-à-dire un ensemble de constituants.

Soit le constituant composé $X = \{X_1, X_2, \ldots, X_p\}$

Si $a_1, a_2, \ldots, a_p$ désignent p valeurs (non forcément distinctes), les p affectations :

$$X_i := a_i \qquad i = 1,2,\ldots,p$$

sont notées : $\langle X : a \rangle = \{\langle X_1 : a_1 \rangle, \langle X_2 : a_2 \rangle, \ldots, \langle X_p : a_p \rangle\}$

Relations n-aires (N-ARY RELATIONS)

Une relation n-aire est définie par 3 éléments :

- la donnée d'un n-uplet de constituants $X = \{X_1, X_2, \ldots, X_n\}$,
- l'attribution d'un champ U_i à chaque constituant X_i,
- un prédicat qui, à toute affectation <X:a> conforme aux champs, donne une réponse exclusivement vraie ou fausse.

Elle est notée : $R[X_1, X_2, \ldots, X_n]$; $X_i :\epsilon U_i$ $\quad i = 1,2,\ldots n$

ou : $R[X]$; $X :\epsilon U$

Entité (ENTITY)

Une entité est une affectation <X:a> pour laquelle l'évaluation du prédicat est vraie, soit :

$$\| R[<X:a>] \| = \text{vrai}$$

Une entité est donc un ensemble d'affectations que l'on notera par abus de langage : $\|R[a]\| = \text{vrai}$

Représentation d'une relation

Une relation formée de constituants élémentaires peut se représenter sous forme tabulaire en donnant une colonne par constituant et une ligne par n-uplet d'informations

R :

X_1	X_2		X_n	CONSTITUANTS
a_1^1	a_2^1		a_n^1	
a_1^2	a_2^2		a_n^2	
				VALEURS (entités)
a_1^k	a_2^k		a_n^k	

La valeur a_i^j correspond à la $i^{ème}$ affectation de la $j^{ème}$ entité.

L'ordre des colonnes est quelconque.

Notion de constituant et fonction d'accès

Dans un tel modèle, on peut interpréter la notion de constituant comme une fonction d'accès qui, à une entité de la relation R, fait correspondre une valeur d'un champ ; le constituant étant une fonction d'accès unidirectionnelle d'un élément de R vers le champ.
(Une colonne d'un tel tableau contient les valeurs prises par un constituant, pour différentes entités, dans le champ qui lui est attribué).

Base de données

C'est une collection de relations, variable dans le temps.

A1-2 - Exemple d'une collection de relations

Cet exemple décrit en partie une base de données concernant la gestion de la scolarité dans un établissement scolaire où l'on a les 6 relations ci-dessous constituant une collection

GESTSC = {ETUDIANT, GROUPE, SALLE, MATIERE, PROF, EMPT}

Champs et constituants :

NOM :∈ NOMPERSONNE,
SALAIRE :∈ REEL,
CDM :∈ CODEMATIERE,
RESUME :∈ TEXTE,
RESP :∈ NOMPERSONNE,
CDS :∈ CODESALLE,
CAPACITE :∈ ENTIER,
NUMERO :∈ NUMETUDIANT,
CDGR :∈ ENTIER,
NBELEV :∈ ENTIER,
HEURE :∈ ENTIER ;

Relations :

SALLE [CDS, CAPACITE] ;
PROF [NOM, SALAIRE, CDM] ;
MATIERE [CDM, RESUME, RESP] ;
ETUDIANT [NUMERO, CDGR] ;
GROUPE [CDGR, NBELEV] ;
EMPT [NOM, CDM, CDGR, CDS, HEURE] ;

La relation SALLE décrit pour chaque salle son code et sa capacité.

La relation PROF donne pour chaque professeur son nom, son salaire et la matière qu'il enseigne.

La relation MATIERE donne pour chaque matière son code, un résumé et le nom d'un professeur responsable pédagogique.

La relation ETUDIANT décrit pour chaque étudiant son numéro et son groupe d'appartenance.

La relation GROUPE définit pour chaque groupe son code et le nombre d'étudiants.

La relation EMPT précise un emploi du temps en donnant par professeur et par groupe la matière, la salle et l'heure de début du cours.

Une réalisation de la relation EMPT est un ensemble d'entités que l'on peut représenter sous forme d'un tableau de valeurs :

EMPT

NOM	CDM	CDGR	CDS	HEURE
dupont	math	1	B114	8
dupont	math	2	B114	10
durand	anglais	4	C108	8
martin	gestion	1	C108	14
dupont	math	4	C108	16
martin	gestion	2	B114	16

A1-3 - Principales opérations algébriques définies sur les relations

Somme et produit

Soient R[X] et S[Y] deux relations dont les constituants communs éventuels sont munis de mêmes champs.

Z = X ∨ Y l'union des constituants.

La somme et le produit des 2 relations sont définis par leur prédicat

$$\|(R+S)[Z]\| \triangleq (\|R[X]\| \text{ ou } \|S[Y]\|)$$

$$\|(R*S)[Z]\| \triangleq (\|R[X]\| \text{ et } \|S[Y]\|)$$

Complément

Soit une relation R[X], son complément est noté $\bar{R}[X]$ et il est défini par son prédicat :

$$\|\bar{R}[X]\| \triangleq \neg \|R[X]\|$$

Projection

Soit R[X,Y] une relation
et $X :\in U$

La projection suivant X de cette relation est la relation au constituant Y notée [Y]R[X,Y], définie par son prédicat :

$$\|[Y]R[X,Y]\| \triangleq \exists\, a\ \|R[a,Y]\|,\ a \in U$$

Soit R[X,Y,Z] une relation telle que $X :\in U$
$Y :\in V$

Si <Y:b> est une affectation, on définit la projection suivant X à partir de b comme la relation au constituant Z notée :

[Z, <Y:b>] R [X,Y,Z]

et définie par son prédicat

$a \in U,\ b \in V,\ \|[Z, <Y:b>]\ R\ [X,Y,Z]\| \triangleq \exists\, a\ \|R[a,b,Z]\|$

La nouvelle relation obtenue par projection est appelée sous-relation de R.

Propriété de la projection : La projection d'une relation ne dépend pas du chemin utilisé :

$$\|[Z]\ R\ [X,Y,Z]\| = \|[Z][X,Z]\ R\ [X,Y,Z]\| = \|[Z][Y,Z]\ R\ [X,Y,Z]\|$$

A1-4 - Formulation de requêtes : exemples

La requête la plus simple est celle qui peut être obtenue par une projection, par exemple pour répondre à la requête R1 :

R1 : Trouver les numéros des étudiants du groupe 4 ; il suffira d'écrire l'expression algébrique :

[NUMERO, <CDGR : 4>] ETUDIANT

ou encore

R2 : Trouver les noms des professeurs de mathématiques qui ont un salaire de plus de 4000 francs

[NOM, <SALAIRE :> 4000>, <CDM : math>] PROF

L'opération de produit apparaît dans les requêtes où l'expression algébrique nécessite l'utilisation de plusieurs relations.

R3 : Trouver le code des groupes qui ont des cours dans des salles de plus de 50 places

[CDGR] (EMPT * [CDS, <CAPACITE :> 50>] SALLE)

Des opérations prédicatives sont aussi définies sur le même modèle relationnel [L10].

Les requêtes peuvent être formulées à l'aide du langage algébrique ou d'un langage prédicatif (le langage alpha).

A1-5 - Relations fonctionnelles et index d'une relation

Relation fonctionnelle

Soit la relation R[X,Y,Z], on dit qu'il existe une relation fonctionnelle entre X et Y (notée $X \overset{R}{\rightarrow} Y$) sous-relation de R, lorsque :

a, b, c et a', b', c' étant des affectations respectives à X, Y, Z, on a :

$\forall a\ \forall b\ \forall b'\ \forall c\ \forall c'$ ||R[a,b,c]|| et ||R[a,b',c']|| => (b ≠ b')

Ainsi, la connaissance de X détermine au plus un seul Y sans avoir besoin de préciser Z (ensemble de constituants éventuellement vide).

Exemple : *COURS [PROFESSEUR, SALLE, HEURE]*

On a la relation fonctionnelle PROFESSEUR, HEURE → SALLE ; car un professeur donné à une heure précise ne peut être que dans une seule salle.

Index d'une relation (Primary key)

Soit la relation R[A] formée sur le n-uplet de constituants
$A = \{A_1, A_2, \ldots, A_n\}$

L'index de la relation R est le constituant, éventuellement composé, I de A, tel que :

$\forall A_j \in A$, il existe la relation fonctionnelle $I \overset{R}{\rightarrow} A_j$

et $\nexists I' \subset I$, tel que $\forall A_j \in A$, il existe la relation fonctionnelle $I' \overset{R}{\rightarrow} A_j$

Propriétés des relations fonctionnelles

* Les relations fonctionnelles ont les propriétés de :
  - transitivité : $\forall E,F,G$ si $E \overset{R}{\rightarrow} F$ et $F \overset{R}{\rightarrow} G$, alors $E \overset{R}{\rightarrow} G$
  - réflexivité : $\forall E \quad E \overset{R}{\rightarrow} E$
  - projection : si $E \overset{R}{\rightarrow} F,G$ alors $E \overset{R}{\rightarrow} F$ et $E \overset{R}{\rightarrow} G$
  - augmentation : si $E \overset{R}{\rightarrow} F$, alors $\forall G \quad E,G \overset{R}{\rightarrow} F$
  - additivité : si $E \overset{R}{\rightarrow} F$ et $E \overset{R}{\rightarrow} G$, alors $E \overset{R}{\rightarrow} F,G$
  - pseudo-transitivité : si $E \overset{R}{\rightarrow} F$ et $F,G \overset{R}{\rightarrow} H$, alors $E,G \overset{R}{\rightarrow} H$

(où nous supposons avoir une relation R formée sur les constituants E, F, G, H).

* Une relation fonctionnelle $X \overset{R}{\rightarrow} Y$ est dite élémentaire :
  - si pour toute partie $X' \subset X$, $X' \overset{R}{\rightarrow} Y$ n'est pas une relation fonctionnelle
  - et si elle n'est pas triviale ($X \overset{R}{\rightarrow} X$ est une relation fonctionnelle dite triviale).

* Soit la relation R[A], $A = \{X,Y,\ldots\}$;
une relation fonctionnelle élémentaire $X \overset{R}{\rightarrow} Y$ est directe :

- s'il n'existe aucun constituant composé T, partie de A, non index, tel que $X \overset{R}{\rightarrow} T \overset{R}{\rightarrow} Y$

  où les relations fonctionnelles $X \overset{R}{\rightarrow} T$ et $T \overset{R}{\rightarrow} Y$ sont élémentaires.

A1-6 - Normalisation et décomposition de relations

La normalisation est un processus de remplacement d'une collection donnée de relations par des relations dont la "structure est plus simple". Cette normalisation est réversible, car on peut retrouver la collection initiale ; elle n'entraîne aucune perte d'informations.

Formes normales de CODD : définitions

Soit une relation R[A]

* 3ème forme :
Une relation R est en 3ème forme si pour tout index I et pour tout constituant X n'appartenant pas à I

$I \overset{R}{\rightarrow} X$ est élémentaire directe

* 2ème forme :
Une relation R est en 2ème forme si pour tout index I et pour tout constituant X n'appartenant pas à I

$I \overset{R}{\rightarrow} X$ est élémentaire

* 1ère forme :
Toute relation R qui n'est ni en 3ème forme, ni en 2ème forme est en 1ère forme.

Lorsque l'on passe de la première forme à la troisième forme, les conditions sont de plus en plus restrictives. On peut remarquer qu'une relation en troisième forme est moins redondante en informations qu'une relation en deuxième ou première forme.

Décomposition d'une relation

L'opération de décomposition d'une relation tend, partant d'une relation en première forme, à la décomposer en des relations de deuxième ou troisième forme , puis en répétant ce processus, on peut aboutir uniquement à des relations en troisième forme.

Une relation R[X,Y,Z] est décomposable en deux relations R1 et R2 si :

$R = R1 * R2$

avec $R1 = [X,Y]R$ et $R2 = [X,Z]R$

Pour cela, il faut et il suffit que [2] :

si $X :\in U$

$\forall x \in U$

$R\ [<X:x>,Y,Z] = R\ [<X:x>,Y] * R\ [<X:x>,Z]$

ANNEXE A2

LE MODÈLE "DATA SEMANTICS" DE J.R. ABRIAL

A2-1 - Les ensembles

A2-2 - Les relations binaires

A2-3 - Comment obtenir de l'information du modèle ?

A2-4 - Extension des sémantiques de base

A2-5 - Ecriture d'algorithmes

A2-6 - Conclusion.

J.R. ABRIAL propose dans [L1] un système de base de données "idéal" pour définir, utiliser, organiser et partager des données par plusieurs utilisateurs, et dans [A1] un formalisme pour décrire la sémantique de la définition et de l'utilisation des données.

Notre présentation est extraite de [A1] et elle résume comment ce modèle permet de décrire (et traiter) des objets par les liaisons binaires qu'ils ont avec d'autres objets.

A2-1 - Les ENSEMBLES (ou CATEGORIES)

Les ensembles sont de deux natures différentes :

- les ensembles concrets dont on doit créer (ou tuer) les éléments
- les ensembles abstraits regroupant les types simples classiques.

* L'opérateur "cat" est utilisé pour indiquer l'existence d'un nouvel ensemble (abstrait ou concret , appelé ASET ou CSET dans [L1]).

```
PROF = cat
MATIERE = cat
SALLE = cat
SALAIRE = cat
```

pour décrire les ensembles des PROFesseurs, des MATIEREs, des SALLEs, et des SALAIREs.

* L'opérateur "generate" sert à créer un élément (un objet) d'un ensemble

```
generate PROF
```

et éventuellement à lui associer :

- . un nom externe discriminant DUPONT = generate PROF
- . ou un sobriquet (une variable le repérant) X ← generate PROF

* L'opérateur "kill" supprime un élément dans un ensemble

```
kill DUPONT
kill X
```

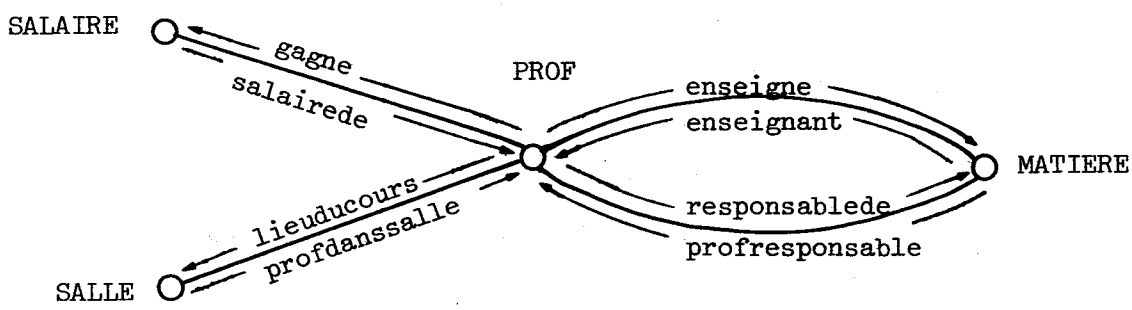
A2-2 - Les relations binaires

Elles permettent de décrire des connexions entre des éléments abstraits ou concrets appartenant à certains ensembles.

* Les opérateurs "rel" et "afn" : servent à définir une nouvelle relation binaire ainsi que les deux fonctions d'accès et les cardinaux des ensembles qu'elles décrivent (ces cardinaux sont les bornes inférieures et les bornes supérieures du nombre d'objets qui peuvent être connectés).

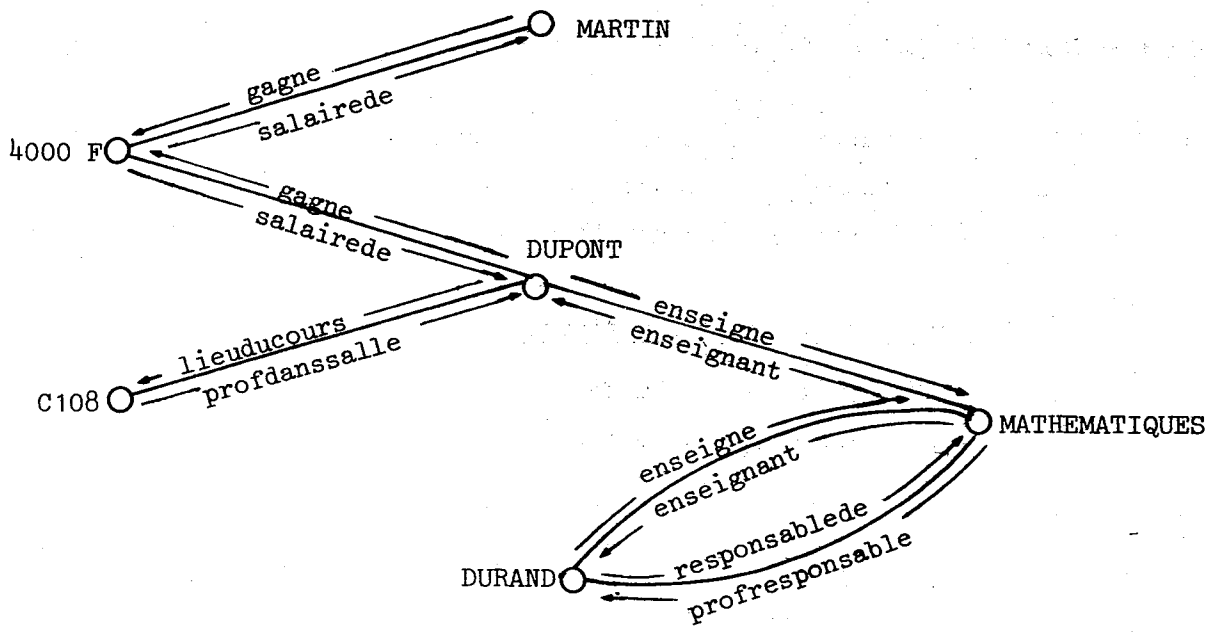
R1 = rel (PROF, SALAIRE, gagne = afn(1,1), salairede = afn(0,∞))
R2 = rel (PROF, MATIERE, enseigne = afn(1,5), enseignant = afn(1,∞))
R3 = rel (MATIERE, PROF, profresponsable = afn(1,1), responsablede = afn(0,5)
R4 = rel (PROF, SALLE, lieuducours = afn(1,1), profdanssalle = afn(1,1))

* Représentation graphique des relations binaires :



* Représentation graphique des réalisations de relations binaires :

- dupont gagne 4000 F et enseigne les mathématiques
- durand enseigne les mathématiques dont il est le responsable pédagogique
- martin gagne 4000 F
- dupont fait cours dans la salle C108



* Les opérateurs ":∋" et ":∌" permettent de créer et supprimer des connexions entre des éléments.

Enseignant(X) :∋ Y veut dire que si X et Y sont respectivement des noms (ou sobriquets) de MATIERE et PROF, Y enseigne X et X a comme enseignant Y.

Si le cardinal maximum d'une fonction d'accès = 1, on utilise le symbole "←"

lieuducours (Y) ← C108

DUPONT = generate PROF MATHEMATIQUES = generate MATIERE enseignant(MATHEMATIQUES) :∋ DUPONT	pour indiquer que deux objets sont reliés entre eux dans la relation R2
DURAND = generate PROF	
enseignant(MATHEMATIQUES) :∋ DURAND	relation R2
profresponsable(MATHEMATIQUES) :∋ DURAND	relation R3
enseignant(MATHEMATIQUES) :∌ DUPONT	destruction d'une connexion

Par exemple, pour exprimer qu'un professeur ne peut être responsable que d'une matière qu'il enseigne (cf. les relations § A-2-2), on écrit :

```
updater (profresponsable) ← prog (X,Y)
                            if X ∈ enseigne (Y)
                               then std (X,Y)
                               else échec
                            end
```

Une instruction profresponsable (a) ← b donne alors le contrôle au programme ci-dessus avec "a" et "b" comme paramètres effectifs, ce programme contrôle si "a ∈ enseigne(b)", si oui, il fait l'action standard de mise à jour (std), ou sinon il retourne une erreur.

A2-5 - Ecriture d'algorithmes

J.R. ABRIAL précise des mécanismes de base pour exprimer des programmes définissant les extensions diverses de sémantiques.

* conditions : if ... then ... end
  if ... then ... else ... end

* boucles : for ... end
  associé à up 1 pour retourner au for
  ou à down 1 pour effectuer une sortie forcée

Par exemple, pour "balayer" les éléments d'une fonction d'accès F, on écrit :

```
for  X ← F(Y)
|    ...
|    up 1
end
```

* programmes :

Un programme est un objet abstrait qui peut avoir un résultat unique (opérateur return) ou multiple (opérateur resume).

Soient les relations :

rel (personne, personne, parent = afn(2,2), enfant = afn(0,∞))
rel (personne, personne, grandparent) = afn(4,4), petitfils=afn(0,∞)).

On peut définir l' "accessor" pour la fonction d'accès "**grandparent**" par :

```
accessor (grandparent) ← prog(X)
                         if ¬ (X is personne) then échec end
                         for Y ← parent (X)
                         |   for Z ← parent (Y)
                         |   |   resume (Z)
                         |   |   up 1
                         |   end
                         |   up 1
                         end
```

Si l'on écrit maintenant

```
for a ← grandparent (b)
|   up 1
end
```

à chaque retour au début de la boucle for, le processus initialisé par la boucle donne comme résultat partiel le prochain grand-parent et s'arrête pour attendre le prochain appel.

A2-6 - Conclusion

Citons J.R. ABRIAL [A1] :

"... the model we have described in this paper is not a language nor a generalized Data Base Management System proposal. We could merely consider it as a specification tool that could (?) help people in writing programs. In this sense, this paper belongs to the "structured programming" area more than to the "Data Base" literature ...".

A3-1 - L'enregistrement (= RECORD)

A3-2 - Le lien (= SET)

A3-3 - Structures de données possibles

A3-4 - Déclarations complémentaires

A3-5 - Les primitives du DML

A3-6 - Conclusions : notions de schéma et de sous-schéma.

ANNEXE A3

LE DDL ET LE DML DE CODASYL

Le "Data Base Task Group" énonce dans [A7] et [A8] un ensemble de recommandations pour un SGBD. Nous ne donnerons que les principales définitions : le lecteur pourra retrouver tous les détails dans les deux rapports cités ci-dessus.

Nous nous sommes servis des documents [A3] et [L25] comme principaux supports pour résumer les deux concepts :

- le DDL (Data Description Language)
- le DML (Data Manipulation Language).

Tout d'abord, trois éléments principaux permettent la structuration des données :

. le DATA-ITEM
. le RECORD
. le SET.

A3-1 - L'enregistrement (= RECORD)

* Un ENREGISTREMENT est composé :
  - de constituants simples (= DATA-ITEM)
  - et/ou de constituants composés (= VECTOR OF DATA-AGGREGATE)
  - et/ou de groupes de constituants (= REPEATING GROUP OF DATA-AGGREGATE).

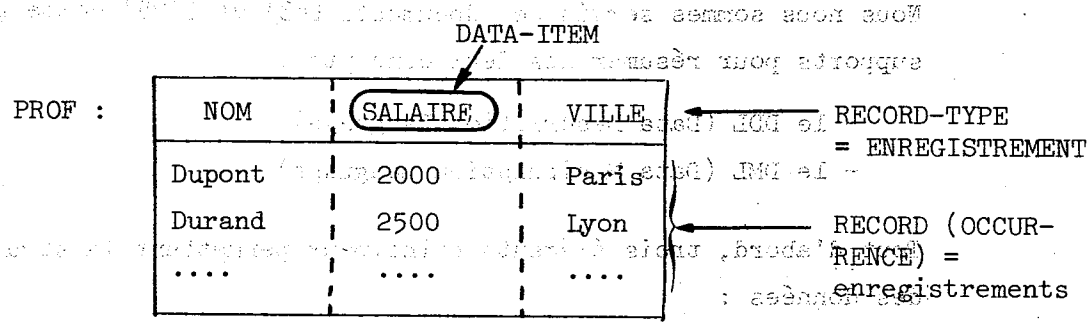
* DATA-ITEM : c'est la plus petite partie d'une donnée nommée. L'occurrence d'un data-item est la représentation d'une valeur.

Exemple de description structurelle d'un ENREGISTREMENT dans le DDL :

```
RECORD NAME IS PROF
   02 NOM TYPE IS CHARACTER 20
   02 SALAIRE TYPE IS FIXED DECIMAL 5
   02 VILLE TYPE IS CHARACTER 10
```

Un ensemble de clauses supplémentaires permettent de préciser diverses caractéristiques des data-items.

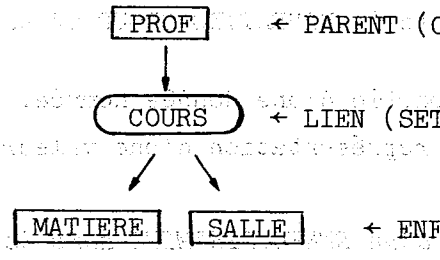
A chaque enregistrement (= RECORD-TYPE, noté ENREGISTREMENT) correspond plusieurs réalisations (= OCCURRENCE, notées enregistrements) dans la base de données.



A3-2 - Le LIEN (= SET)

Toutes les liaisons entre ENREGISTREMENTS sont faites par des LIENS.
Un LIEN est une relation entre un ENREGISTREMENT (= RECORD-TYPE) déclaré comme PARENT (= OWNER) du LIEN et un ou plusieurs ENREGISTREMENTS (= RECORD-TYPE) déclarés comme ENFANTS (= MEMBER).

Un LIEN peut être représenté graphiquement ; exemple :



Ceci peut sommairement se traduire dans le DDL par :

```
SET NAME IS COURS
    OWNER IS PROF
    MEMBER IS MATIERE
    MEMBER IS SALLE.
```

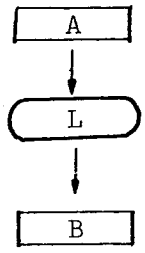
Du point de vue des données, à un LIEN peuvent correspondre plusieurs réalisations (= OCCURRENCE, notées en minuscules liens) de ce LIEN (en majuscules).

Chaque lien contient une réalisation de son PARENT et un nombre quelconque de réalisations d'un ENFANT : il s'agit donc d'une relation de type 1:n.
Un lien l de l'exemple précédent peut s'écrire :
l = (dupont, (algèbre, proba, ...), (C108, B114, ...)).

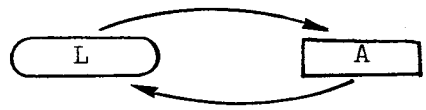
A3-3 - Structures de données possibles

Cette notion de LIEN est l'élément de base qui permet la description des structures de données, aussi est-il important d'en étudier toutes les possibilités.

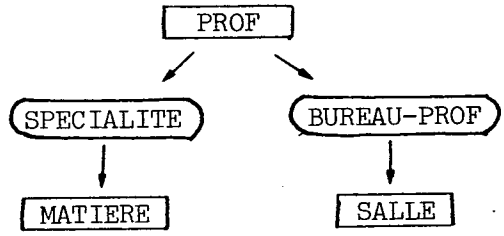
1. Un LIEN doit avoir un PARENT (RECORD-TYPE) et au moins un ENFANT (RECORD-TYPE) ; et un ENREGISTREMENT (RECORD-TYPE) ne peut pas être PARENT et ENFANT d'un même LIEN.
   Structure de base autorisée :



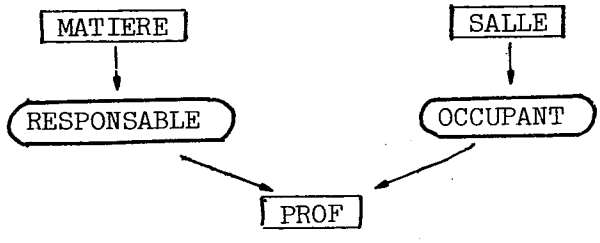
structure interdite :



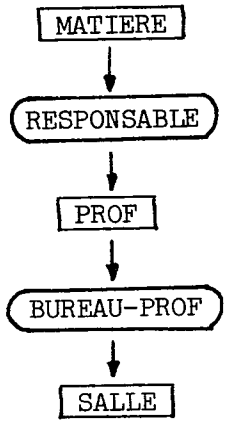
2. Tout ENREGISTREMENT peut être PARENT de plusieurs LIENS différents (structure d'arbre)



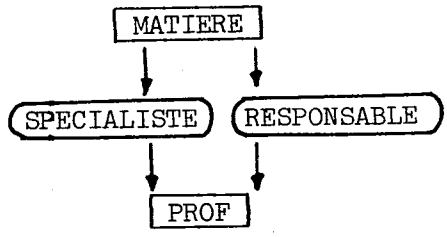
3. Tout ENREGISTREMENT peut être ENFANT de plusieurs LIENS différents.



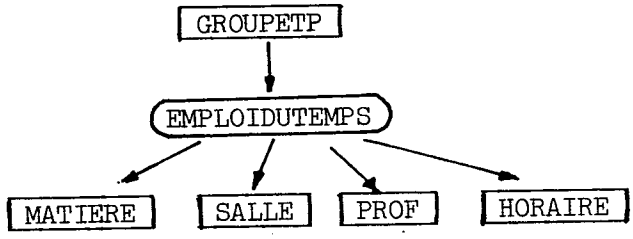
4. Tout ENREGISTREMENT peut être à la fois :
   - ENFANT d'un ou plusieurs LIENS différents
   et - PARENT d'un ou plusieurs LIENS différents



5. Deux ENREGISTREMENTS peuvent être reliés entre eux par un nombre quelconque de LIENS différents.



6. Un LIEN ne peut avoir qu'un ENREGISTREMENT comme PARENT, mais il peut avoir plusieurs ENREGISTREMENTS différents comme ENFANTS.



CODASYL permet ainsi la structure-réseau (remarques ci-dessus), et n'interdit pas l'existence de circuits (il met en garde l'utilisateur éventuel de structure avec circuit à cause de la cohérence des informations et du risque de "dead-lock" pour certaines opérations).

A3-4 - Déclarations complémentaires

Un ENFANT (= MEMBER) d'un LIEN (= SET) peut être déclaré comme
- automatique (= MEMBER AUTOMATIC) ou non (= MEMBER MANUAL)

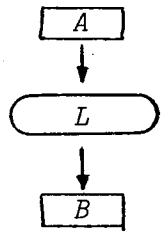
ou - obligatoire (= MEMBER MANDATORY) ou non (= MEMBER OPTIONAL)

ou - obligatoire et automatique (= MEMBER MANDATORY AUTOMATIC)

Ces déclarations ont des répercussions sur les mécanismes de création et d'élimination des enregistrements et des liens (cf. A3-5).

Exemple :

* *Ainsi, cette structure :*



avec la déclaration : SET NAME IS L
OWNER IS A
MEMBER IS B MANDATORY

signifie qu'une réalisation de B ne peut exister dans la base que si son PARENT existe ; si l'on détruit une réalisation a de A, il faut détruire automatiquement le lien auquel appartient a et toutes les réalisations b de B correspondantes.

* *Si l'on a déclaré SET NAME IS L*
OWNER IS A
MEMBER IS B AUTOMATIC

alors l'ajout d'une réalisation b de B est suivi automatiquement de la mise à jour du lien auquel elle appartient (dans le cas MEMBER MANUAL, la mise à jour du lien doit être prévue explicitement par le programmeur).

Bien sûr, d'autres clauses complémentaires permettent de préciser diverses caractéristiques des ENREGISTREMENTS et des LIENS, en particulier les descriptions de chemins d'accès (voir [6] et [7]).

Nous indiquons simplement que pour certains ENREGISTREMENTS, il est possible de déclarer dans la méthode d'accès un constituant discriminant (qui peut être un constituant composé) avec la clause LOCATION MODE.

Exemple : *Si l'on veut accéder aux réalisations de PROF en utilisant la valeur du NOM, et que l'on interdit la présence de deux noms identiques dans la base, on écrit :*

```
RECORD NAME IS PROF
        LOCATION MODE IS CALCULATING USING NOM
        DUPLICATES NOT ALLOWED
        NOM
        SALAIRE
        .....
```

A3-5 - Les primitives du DML

Sept primitives principales constituent l'essentiel du langage de manipulation en CODASYL ; mais il faut d'abord préciser deux notions importantes qui s'y rattachent :

1) La notion de zone de travail utilisateur (USER WORK SPACE) :
c'est une zone intermédiaire entre l'utilisateur et la base de données, qui contient un emplacement pour chaque constituant de chaque enregistrement.
Elle correspond à la DATA DIVISION d'un programme COBOL.
En entrée, les valeurs des constituants d'un enregistrement à stocker dans la base sont rassemblées dans cette zone avant d'être envoyées à leur emplacement définitif.
En sortie, la recherche d'une réalisation particulière permet de repérer un emplacement dans la base, les valeurs correspondantes n'étant accessibles qu'après transfert dans la zone de travail.

2) La notion d'enregistrement (ou lien) courant (CURRENCY)
La réalisation d'un ENREGISTREMENT (RECORD-TYPE)(ou d'un LIEN (SET)) à laquelle le programme actif a accédé en dernier est la réalisation courante (d'enregistrement ou de lien) ; elle est désignée par EC dans ce qui suit.

Présentation logique des primitives :

* TROUVER (= FIND) : "Trouver a", pour localiser dans la base une réalisation a d'enregistrement et la rendre EC pour le programme actif. L'enregistrement cherché peut être spécifié par la valeur d'un de ses constituants, valeur donnée dans la zone de travail.

* TRANSFERER (= GET) : transférer de la base de données vers la zone de travail l'EC.

* MODIFIER (= MODIFY) : mise à jour de l'EC.

* RAJOUTER (= INSERT) : rajouter l'EC dans une ou plusieurs réalisations de liens. "Le rajout de a dans le lien l de L" nécessite que a soit un ENFANT de l pour être exécuté.

* ENLEVER (= REMOVE) : enlever l'EC d'une ou plusieurs réalisations de liens. "enlever a du LIEN L", cette commande provoque l'élimination de l'appartenance de a à la réalisation correspondante du LIEN L ; elle nécessite que a soit une réalisation d'un ENFANT (non MANDATORY) du LIEN L.

* STOCKER (= STORE) : créer une nouvelle réalisation d'enregistrement et la rendre EC. La commande "stocker a" provoque :
  - la création dans la base de l'enregistrement a
  - la création dans la base de liens dont a est le parent (car a est une réalisation d'un enregistrement PARENT de ces liens)
  - le rajout de a dans les liens dont a est un MEMBER AUTOMATIC.

* DETRUIRE (= DELETE) : détruire l'EC de la base.
  La commande "détruire a" provoque :
  - la destruction de l'enregistrement a de la base de données
  - l'élimination de a de tous les liens dont il est ENFANT
  - la destruction de tous les liens dont a est ENFANT

- la destruction des enregistrements (MEMBER MANDATORY) de lien dont a est un PARENT et ceci de manière récursive.

Ces primitives comportent différentes options non étudiées ici.

A3-6 - Conclusions : notion de schéma et de sous-schéma

Terminons cette présentation par deux notions importantes dans le DDL de CODASYL : le SCHEMA et le SUB-SCHEMA. La séparation de ces deux concepts permet la partition de la description logique de la base de données. Un programme n'est, certes, pas concerné par la totalité de la base et seule la description de son univers par le SUB-SCHEMA lui est nécessaire. Par contre, le SCHEMA décrit dans sa totalité les LIENS, ENREGISTREMENTS et CONSTITUANTS qui composent la base de données (un SUB-SCHEMA est décrit pour un programme COBOL, avec des instructions spéciales en COBOL).

Pour conclure, nous mentionnerons que le DDL de CODASYL comprendaussi de nombreuses clauses concernant l'implantation physique et les chemins d'accès aux données.

ANNEXE B

LE PROJET MACSI-1

B1 - Le modèle de description fonctionnelle

B2 - Etude de la description modulaire

B3 - Réflexions méthodologiques

B4 - Langage de description de modules.

Nous présentons les principaux concepts du modèle MACSI-1 introduits dans [L22] et [L5].

B1 - LE MODELE DE DESCRIPTION FONCTIONNELLE D'UN PROJET INFORMATIQUE : APPROCHE QUALITATIVE

Pour décrire la structure du système d'informations dont la conception est l'enjeu d'un projet informatique, il nous faut un modèle de ce système.

Par SYSTEME, nous entendons communément un ensemble de constituants ayant à la fois leurs caractéristiques propres et des relations entre eux, le fonctionnement du système dépendant tout autant, sinon plus, des relations entre les constituants que de leurs caractéristiques intrinsèques. Le terme de "système" a, de plus, une connotation dynamique dans la mesure où l'on souhaite expliquer par sa description l'évolution dans le temps de ce système.

Pour proposer un modèle suffisamment général de système d'informations permettant de représenter les propriétés fonctionnelles intéressantes de la plupart des applications informatiques de gestion, il faut donc dégager les types de constituants de base qui apparaissent dans ce genre d'applications et la nature des relations signifiantes par lesquelles ils sont interconnectés.

Une expérience même très sommaire, en informatique de gestion, laisse souhaiter que ce modèle permette de décrire :

- les opérations automatiques de traitement de l'information dans l'application informatique proprement dite,
- les opérations qui, dans le système d'organisation de l'entreprise ou de l'administration, ont un lien direct avec les opérations informatiques,
- l'enchaînement dynamique de ces deux types d'opérations pour constituer des fonctions complexes qui sont associées aux finalités du projet informatique.

Essayons, avec l'aide de la figure 1, de préciser les constituants de base et leurs relations qui permettront de représenter l'application informatique, son organisation environnante et l'enchaînement dynamique des opérations dont elle est constituée.

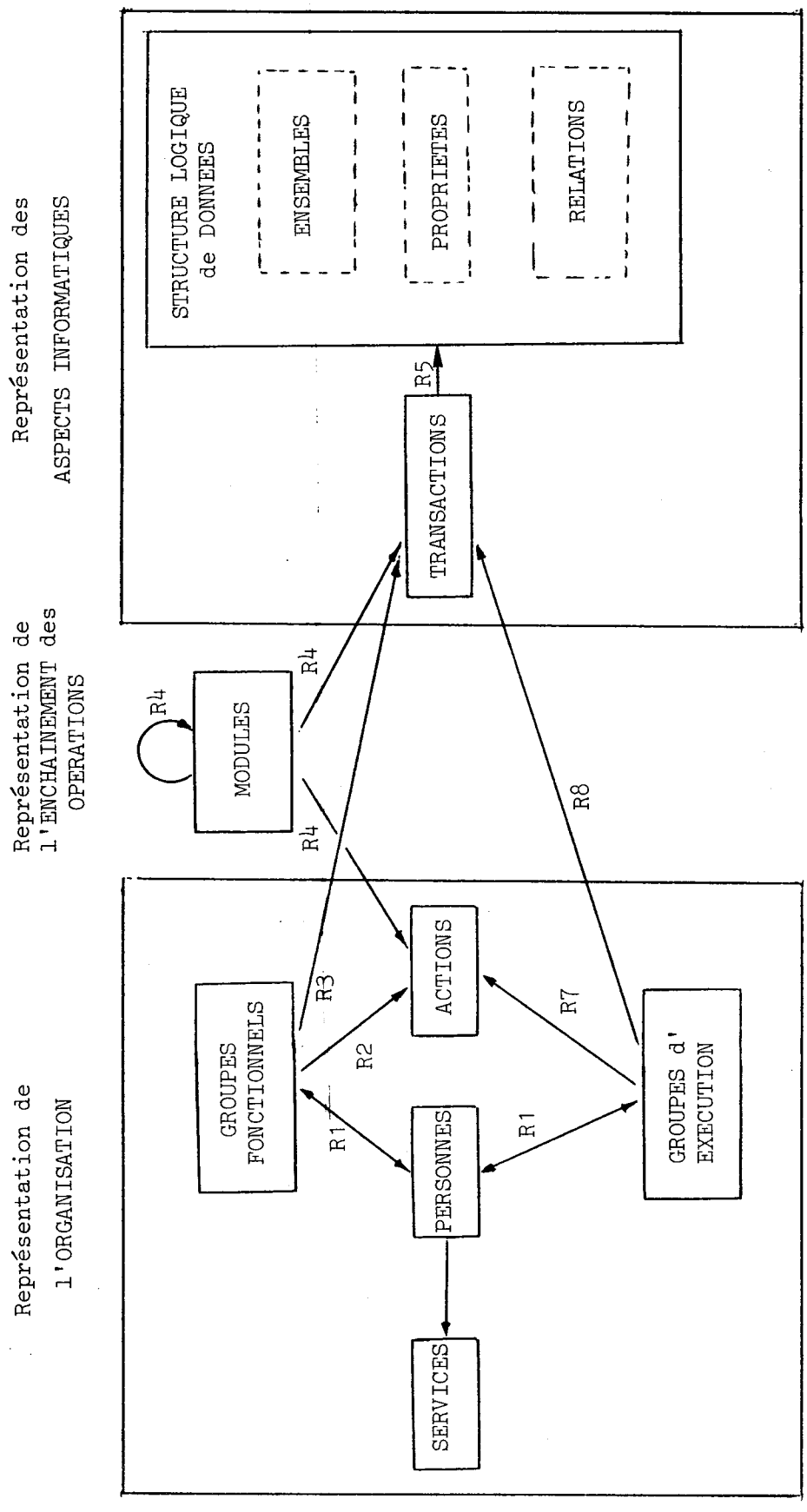


Figure 1 - Les divers constituants d'une application informatique

B1-1 - L'organisation environnante

Nous appellerons *ACTIONS* toutes les opérations qui ont un lien plus ou moins étroit avec le fonctionnement du système d'informations. Sous ce terme générique très vague, sont regroupées des décisions, des opérations de fabrication, des opérations de contrôle technique ou financier et des activités de traitement de l'information non automatisées.

Ces actions sont conçues et réalisées par des *PERSONNES*. Celles-ci ne travaillent pas isolément mais sont solidaires, au niveau de leurs tâches de conception ou d'exécution, à l'intérieur de *GROUPES* (R1). Pour des questions d'administration générale, chaque personne est rattachée à un *SERVICE* (R6) dans l'organigramme de l'entreprise ou de l'administration, service qui peut s'appeler tout aussi bien département, unité, division, etc ...

Parmi ces personnes, il y a les futurs utilisateurs du système d'informations.A ce titre, elles sont concernées non seulement par les actions qu'elles assurent dans l'organisation, mais aussi par les opérations de traitement de l'information qui seront exécutées, à leur initiative, par le service informatique. Nous appellerons *TRANSACTIONS* les opérations élémentaires de traitement automatique de l'information qui sont demandées puis utilisées par les personnes de l'organisation.

Il semble souhaitable de différencier deux types de groupes de personnes dans l'organisation d'un système d'informations.

- Les *GROUPES FONCTIONNELS* ont la responsabilité de la conception des ACTIONS (R2) à prendre en compte dans l'organisation où va s'insérer l'application informatique, et des TRANSACTIONS (R3) de traitement automatisé de l'information.

- Les *GROUPES D'EXECUTION* auront la responsabilité, au moment où le projet sera réalisé et disponible, soit de lancer l'exécution d'un programme correspondant à l'implémentation d'une TRANSACTION (R8), soit d'accomplir une ACTION (R7).

Il est nécessaire d'introduire cette distinction entre les groupes chargés de la conception et ceux chargés de l'exécution car, dans la réalité, pour une tâche donnée, ces deux groupes se confondent rarement. Ainsi, l'auteur d'une lettre est fonctionnellement responsable de sa

conception et sa secrétaire est techniquement chargée de sa réalisation matérielle. De même, un responsable des ventes définit la forme des bons de sortie du stock qu'il gère, mais c'est un magasinier qui assurera l'action de les remplir.

Evidemment, une personne qui appartient à un ou plusieurs groupes fonctionnels au moment de la conception d'un projet peut faire partie d'un ou plusieurs groupes d'exécution lorsque ce projet sera opérationnel.

Précisons aussi pourquoi nous distinguons les "groupes" des "services". En effet, un groupe fonctionnel ou d'exécution est défini par rapport aux seules transactions et actions prises en compte dans le cadre du projet informatique, alors que les services ou départements d'une organisation ont des définitions de fonction qui recouvrent aussi des domaines d'activité complètement extérieurs au cadre du projet. Aussi un groupe est-il le plus souvent :

- soit une partie d'un service limitée aux seules personnes choisies pour assurer certaines responsabilités dans l'application informatique,
- soit un ensemble de personnes issues de plusieurs services qui, pour ce projet informatique, ont le même rôle de conception ou d'exécution.

B1-2 - L'application informatique

Par ses futurs utilisateurs, elle est perçue comme un ensemble de TRANSACTIONS, opérations automatiques de traitement de l'information, en entrée ou en sortie du service informatique considéré comme une boîte noire. Au début d'un projet, la logique d'une TRANSACTION sera en général spécifiée de façon approximative par un groupe fonctionnel d'utilisateurs sous forme de texte libre.

Il restera alors aux informaticiens, face à un ensemble de TRANSACTIONS, à préciser ensuite complètement la logique qui sera automatisée dans un formalisme sans ambiguïté.

La *STRUCTURE LOGIQUE DES DONNEES* définit les relations entre les informations sur lesquelles travaillent les TRANSACTIONS. Cette structure logique des données est distincte des structures de fichiers ou de bases de données qui seront retenues au niveau organique pour la programmation du projet. La relation (R5) entre les TRANSACTIONS et la structure logique des données est définie complètement par la syntaxe qui sera adoptée pour formaliser la logique des TRANSACTIONS qui exploitent ces données.

B1-3 - L'enchaînement dynamique des opérations

TRANSACTIONS et ACTIONS ne s'exécutent pas dynamiquement dans n'importe quel ordre. Les *MODULES* sont des assemblages (R4) d'actions et de transactions, voire de modules ; les règles d'assemblage des constituants d'un module spécifient dans quel ordre ces constituants seront exécutés.

Ces règles doivent par ailleurs forcer au découpage d'un module en éléments plus simples, pour permettre de maîtriser la complexité de l'application toute entière (considérée elle-même comme un module) en la décomposant en constituants ayant une finalité et une structure plus élémentaires.

Ce souci de réduction de la complexité, par une analyse dite descendante du projet, orientera la définition des modules.

B2 - ETUDE DE LA DESCRIPTION MODULAIRE

Un MODULE est une structure composée de TRANSACTIONS, d'ACTIONS et même de MODULES.

B2-1 - Caractéristiques essentielles d'un module

Examinons la structure d'un module décrite en SOCRATE :

```
entité 50 MODULE
    début
    |   CODEMO mot discriminant
    |   DEFINITION texte 10
    |   RESPONSABLE référence une PERSONNE
    |   entité 50 ELEMENTS
    |       début
    |       |   CODELT mot
    |       |   EXECUTANT référence un GROUPE
    |       |   entité 10 SUITE
    |       |       début
    |       |       |   ELEM-SUITE référence un ELEMENT de un MODULE
    |       |       |   CONDITION texte 10
    |       |       fin
    |       fin
    fin
```

Chaque module a un code discriminant (CODEMO) pour l'identifier.

Initialement, il doit être décrit globalement et qualitativement sous forme d'un texte libre (DEFINITION). La notion de MODULE dans MACSI-1 doit permettre de représenter l'enchaînement dynamique des transactions et des actions. La personne d'un groupe fonctionnel (RESPONSABLE) qui décrira la structure du module sera aussi chargée ensuite de décrire dans le détail les actions et transactions constituant le module.

Ces différents constituants d'un module s'appellent ses ELEMENTS. Chaque élément est repéré par son code (CODELT) qui est le code de l'action, de la transaction ou du module auquel il correspond.

Le groupe d'exécution (EXECUTANT) d'un élément de l'action ou de la transaction correspondante doit être indiqué.

B2-2 - Règles de composition d'un module

Règle 1 - Séquence simple : Si un élément A est suivi d'un élément B, B sera déclaré comme un élément-suite (ELEM-SUITE) de la (SUITE) de A. Il n'y a aucune condition particulière pour activer B après A, alors (CONDITION) sera indéfini.

Règle 2 - Choix simple : Un élément A est suivi de B si la condition CO1 est vérifiée, ou de C si la condition CO2 l'est. B et C seront deux réalisations de l'entité SUITE de A ; la première aura dans (CONDITION) le texte définissant CO1, la seconde le texte définissant CO2.

- Comme CO1 et CO2 s'expriment par des textes, il faut préciser que CO2 n'est pas forcément la négation de CO1.

Règle 3 - Processus indépendants : Un élément A est suivi de B et de C qui s'exécuteront en parallèle indépendamment l'un de l'autre : alors A aura deux réalisations de l'entité SUITE, la caractéristique CONDITION pour ces deux réalisations étant indéfinie.

Plus généralement, un élément A peut avoir n réalisations de l'entité SUITE $S_1 \ldots S_i \ldots S_n$. Quand une réalisation S_i est conditionnée par un texte, CO_i, il exprime dans quelle situation S_i peut être activé collatéralement avec $S_1 \ldots S_n$ à la suite de A.

Règle 4 - Chaque module n'a qu'une seule ENTREE correspondant à la première réalisation de l'entité ELEMENT, et plusieurs SORTIES : ce sont tous les éléments qui ont une SUITE correspondant à l'élément fictif SORTIE (CODELT = SORTIE).

L'exécution d'un module est considérée comme débutant avec celle de son unique élément d'entrée.

L'exécution d'un module est considérée comme terminée lorsque celle de tous ses éléments de sortie l'est.

B2-3 - Exemple

Prenons un exemple simple d'application informatique pour la décrire suivant une structure de module : nous essayerons d'en dégager quelques règles méthodologiques supplémentaires.

L'exemple décrit une partie d'une application ayant pour objectif de gérer sur ordinateur les inscriptions à un grand congrès et la comptabilité de celui-ci. Toute l'application elle-même est considérée comme un module ayant comme code M1.

L'entrée unique de M1 est M2, ses sorties M9 et T2.

De même, l'entrée du module M4 est A6, les sorties de M4 sont A10, T6 et T1.

MODULE CODE(1)	Liste (1) des ELEMENTS			Liste (2) des SUIVANTS	
	EXECU-TANT	CODE(2)	Définition de cet élément	ELEM SUI-TE, CODE(3)	CONDITION (éventuellement)
entrée	=	M2	Détermination du programme scientifique	M3	Un an à l'avance
	=	M3	Appel aux communications	A1	
	RESPON	A1	Fixation définitive du programme	A2 A3	
	TECH	A2	Edition du programme	A5	
	TECH	A3	Dessin pour l'imprimeur de la fiche d'inscription	A4	
	IMP	A4	Impression de la fiche d'inscription	A5	
	TECH	A5	Envoi des bulletins d'inscription	M4	
	=	M4	Procédure périodique d'enregistrement des inscriptions	M4 T1	Jusqu'à une semaine de l'ouverture : tous les 15 jours. Deux jours avant le congrès
	TECH	T1	Edition des listes des participants inscrits, triées par : - nationalité - ordre alphabétique - manifestation - jour d'arrivée - hôtel	M5	
	=	M5	Journée d'ouverture - dernières inscriptions. Régularisation des payements	M9 T2	- Au plus tôt, un mois après le congrès, au plus tard six mois. - Si la demande en est faite par le responsable des congrès
	=	M9	Bilan post-congrès	sortie	Si les comptes sont tous soldés
in M1	RESPON	T2	Nouveau tirage des listes, compte tenu des participants réels	sortie	Si les comptes sont tous soldés

Structure modulaire (début)

	Liste (1) des ELEMENTS			Liste (2) des SUIVANTS	
	EXECU-TANT	CODE(2)	Définition de cet élément	ELEM SUI-TE CODE(3)	CONDITION (éventuellement)
M4 entrée	SEC	A6	Réception des bulletins d'inscription remplis	A7 A8	
	PERF	A7	Perforation de ces bulletins	T3	
	TECH	T3	Mise à jour du fichier des inscrits	T4 T5 T6 T1	En cas d'erreur dans le bulletin En cas de trop payé et de payement insuffisant Facultatif, uniquement à la demande du secrétariat
	auto	T4	Edition d'un message d'anomalie	A7	
	auto	T5	Edition d'une demande de payement du solde	A10	
	TECH	A8	Réservation de l'hébergement demandé	A10	
	TECH	A10	Envoi du reçu, de l'avis d'hébergement et éventuellement d'une demande de payement du solde	sortie	
	auto	T6	Edition de la situation des payements	sortie	
fin M4	SEC	T1	(Déjà vu dans M1)	sortie	

Liste des groupes d'exécution distingués dans la structure modulaire :

CODE	DEFINITION
RESPON	Groupe de personnes responsable de l'organisation générale du congrès
TECH	Groupe chargé de l'organisation matérielle du congrès
IMP	Cellule chargée de limpression de la documentation publicitaire du congrès
SEC	Secrétariat chargé du suivi des inscriptions
PERF	Groupe de perforation

Par cet exemple, nous pouvons illustrer la difficulté de présenter sans ambiguïté avec les seules notions précédentes, l'enchaînement tantôt séquentiel, tantôt parallèle, de plusieurs processus.

Si nous représentons sous forme graphique le début de M1, est-ce la situation de la figure 5.1 ou celle de la figure 5.2 qui a été définie dans le tableau précédent ? Autrement dit :
Y a-t-il deux envois différents A5 de documentation aux clients, l'un après A2, l'autre après A4 (cas de la figure 5.1) ou bien (figure 5.2) y a-t-il un envoi unique, une fois terminées à la fois A2 d'une part et la séquence A3, A4 d'autre part ? Le bon sens indique qu'il s'agit du cas de la figure 5.2.

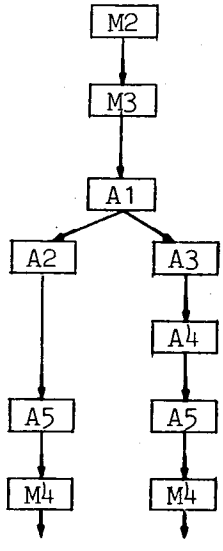


Figure 5.1

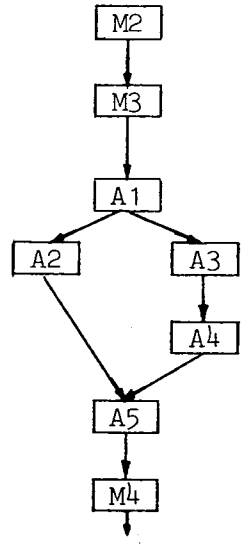


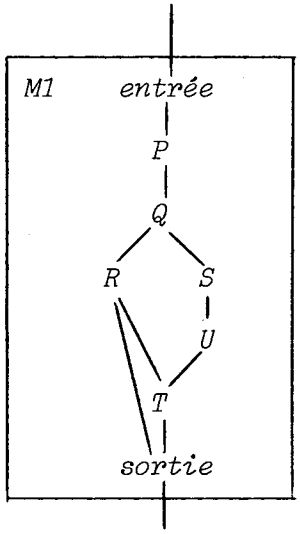
Figure 5.2

Cette difficulté de distinguer les deux cas nous amène à proposer des règles supplémentaires de description de modules.

B2-4 - Règles complémentaires de composition dans un module

Règle 5 - Si deux éléments A et B ont un même suivant C, NON ENCORE DECLARE, cela signifie que C s'exécute une seule fois, quand les exécutions de A et B sont terminées.

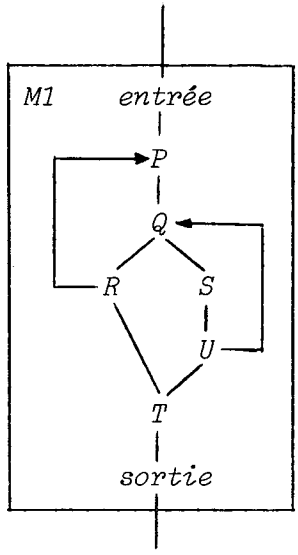
Exemple :



Module	*Elément*	*SUIVANTS*
M1	*entrée* *P*	*Q*
	Q	*R, S*
	R	*T, sortie*
	S	*U*
	U	*T*
	T	*sortie*
fin M1		

Règle 6 - Si un élément A a comme suivant un élément C, DEJA DECLARE, cela signifie qu'il y a une boucle constituée par l'exécution, après A, de celle de C. Cette nouvelle exécution de C est alors suivie de celles des suivants de C.

Exemple :

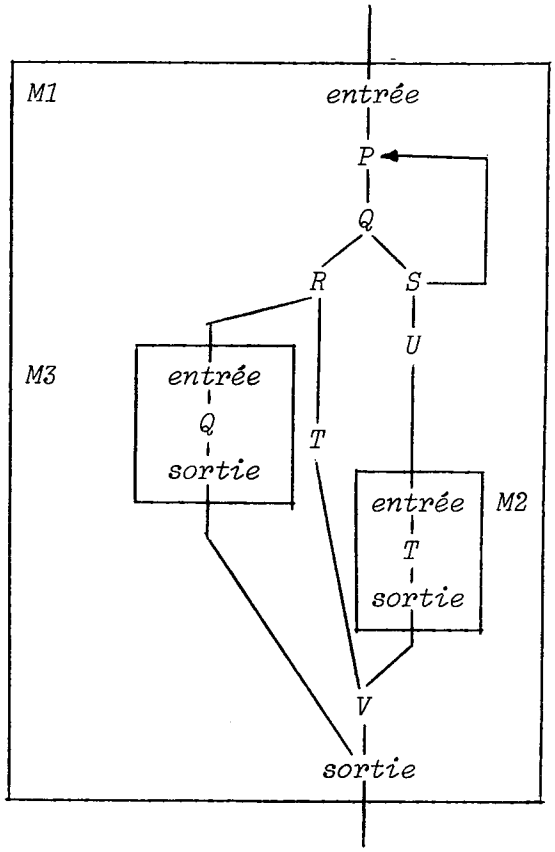


Module	*Elément*	*SUIVANTS*
M1	*entrée* *P*	*Q*
	Q	*R, S*
	S	*U*
	U	*Q, T*
	Q	*R*
	R	*P, T*
	T	*sortie*
fin M1		

Règle 7 - Conséquence - Si après un élément A a lieu une première exécution de C et après un élément B une autre exécution de C, pour pouvoir distinguer ce cas de celui présenté dans la règle 5, une des exécutions de C sera artificiellement présentée comme l'exécution d'un module contenant C.

Aussi, ne doivent pas figurer dans un module deux éléments distincts correspondant soit à une même action, soit à une même transaction ou à un même module.

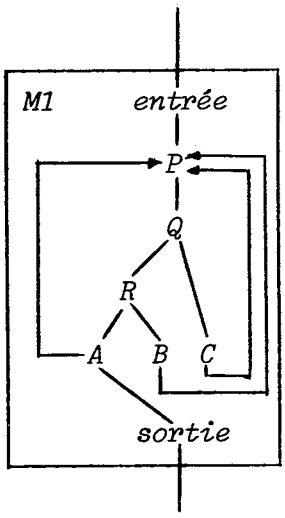
Exemple :



Module	*Elément*	*SUIVANTS*
M1	*entrée*	*P*
	P	*Q*
	Q	*R, S*
	R	*M3, T*
	S	*U, P*
	U	*M2*
	M2	*V*
	T	*V*
	M3	*sortie*
	V	*sortie*
fin M1		
M2	*entrée T*	*sortie*
fin M2		
M3	*entrée Q*	*sortie*
fin M3		

Règle 8 - Pour représenter une boucle correspondant à l'exécution unique d'un élément X, une fois les exécutions de plusieurs autres éléments A, B, C toutes terminées, il faut créer artificiellement un module dont A, B, C sont les sorties.

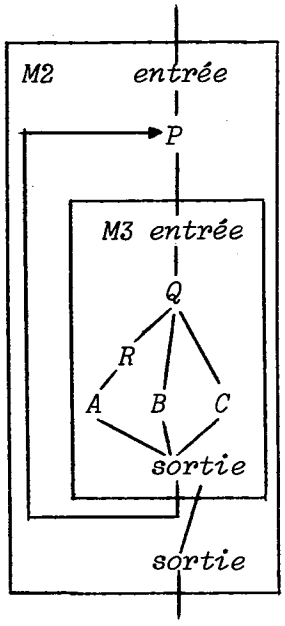
Exemple :



Module	*Elément*	*SUIVANTS*
M1	*entrée P*	*Q*
	Q	*R, C*
	B	*P*
	C	*P*
	R	*A, B*
	A	*P, sortie*
fin M1		

Le module M1 représenté signifie :

"Après A ou après B ou après C, refaire P"



Module	*Elément*	*SUIVANTS*
M2	*entrée P*	*M3*
	M3	*P, sortie*
fin M2		
M3	*entrée Q*	*R, B, C*
	R	*A*
	A	*sortie*
	B	*sortie*
	C	*sortie*
fin M3		

Tandis que le module M2 représenté signifie :

"Une fois que M3 est terminé et donc une fois que A et B et C sont terminés, refaire P"

Remarque :

Il faut donc constater que l'obligation de décrire précisément l'enchaînement dynamique des transactions, actions et modules entraîne quelquefois l'obligation de créer, en application des règles 7 et 8, des modules qui, fonctionnellement, sont artificiels.

B3 - REFLEXIONS METHODOLOGIQUES

Les règles précédentes de définition de la structure d'une application informatique sous forme de MODULES, dont les composants les plus élémentaires sont des ACTIONS et des TRANSACTIONS, appellent quelques remarques sur les possibilités et les limites qu'elles présentent.

I) Il est évident que *l'analyse d'un projet sous forme de modules permet une analyse descendante de celui-ci*, c'est-à-dire une construction du projet en posant d'abord les fonctions macroscopiques (les modules) décomposées de plus en plus jusqu'à leurs détails les plus fins (les actions et les transactions). La possibilité d'une construction par analyse descendante est un objectif commun à toute méthode qui propose une construction modulaire d'un système (cf. en programmation structurée réf. [L6][L7][L19] ou la méthode ISDOS réf. [A25][A24]).

Expérimentalement, le problème reste entier de savoir à quel niveau de finesse la décomposition doit être faite. Nous n'avons défini ici aucun critère permettant de discerner si des éléments déclarés "action" ou "transaction" ne sont pas en fait des processus complexes qui devraient être déclarés comme "modules" et faire ensuite l'objet d'une décomposition. Réciproquement, il se peut fort bien qu'un élément déclaré "module" ne soit pas décomposable en éléments plus détaillés lorsque l'analyse de son fonctionnement sera approfondie.

S'il existe, en programmation structurée, un niveau de finesse maximum de décomposition qui est celui de l'instruction, les critères de décomposition d'un programme en éléments ne peuvent être exprimés qu'en termes qualitatifs : la décomposition d'un programme en modules est laissée finalement à l'appréciation de son auteur.

De même, ici, nous ne pouvons pas définir de critères exacts de bonne décomposition qui permettraient des contrôles automatiques. Tout au plus peut-on définir quelques critères qualitatifs permettant de distinguer si un élément est soit un module, soit au contraire une action ou une transaction. Ils sont résumés dans le tableau suivant :

QUESTION	Réponse selon la nature de l'élément	
	MODULE	ACTION ou TRANSACTION
Y a-t-il unicité du lieu d'exécution ?	pas toujours	si possible
La période d'exécution est-elle bien définie ?	pas toujours	oui
Peut-on affecter un groupe d'exécution à l'élément ?	non	oui
L'élément se présente-t-il comme la répétition d'une opération plus élémentaire ?	quelquefois (alors représenter la boucle si possible)	quelquefois (mais la représentation d'une boucle ne sert à rien)

II) Dans la description d'un module, il est proposé une décomposition en ACTIONS et TRANSACTIONS avec des règles d'enchaînement entre celles-ci, mais entre ces opérations *il n'existe aucune interface analogue aux variables d'états retenues dans d'autres modèles de représentation de systèmes.*

Mais dans MACSI-1, les interfaces existant entre actions et transactions, telles qu'elles sont définies, peuvent être de natures très diverses :

Nature de l'interface	Entre
Un produit	Deux actions de production
Un document	Deux actions de traitement de l'information non automatisées Une action de perforation et une transaction de saisie de données Une transaction d'édition et une action de contrôle
Un fichier Une base de données	Deux transactions
Un rapport	Une action de contrôle et une action de décision
Le plan de fabrication d'une pièce	Une action de conception et une action de fabrication

Cette diversité dans les interfaces possibles est une difficulté qui doit être néanmoins résolue pour qu'au démarrage de l'analyse organique les *volumes d'activité* des opérations puissent être mesurés par le nombre d'unités d' "objets-interfaces" présents dans les flux en entrée et en sortie de ces opérations (nombre de pièces fabriquées, nombre de documents perforés, nombre d'enregistrements d'un fichier, etc). Aussi, les extensions de MACSI-1 pour l'analyse organique, présentées par CHELMINSKI [L5], font-elles apparaître les notions de documents et de fichier : le lecteur pourra s'y reporter. Dans un souci de clarté de présentation des fondements de MACSI-1, nous n'évoquerons pas ici ces possibilités.

III) *Distinction entre actions et transactions*

Il peut sembler prématuré de distinguer, au niveau de l'analyse fonctionnelle, des transactions et des actions de traitement manuel de l'information. Ce choix est, par nature, organique puisqu'il exprime une préférence entre différents moyens techniques, automatiques ou non, pour assurer une fonction de traitement de l'information.

L'expérience prouve que, dès le début d'une étude, des décisions sont prises, même incomplètement, sur quelques options organiques fondamentales. Il est, par exemple, évident, dès le départ d'un projet, que des volumes de données considérables doivent être manipulés par un ordinateur. En conséquence, il est naturel de préciser la nature de certains éléments en tant que "transaction" ou "action" si le bon sens l'exige.

Par contre, il est bien évident que ce genre de décision doit être toujours considéré comme provisoire et susceptible de modification jusqu'au démarrage de l'analyse organique.

IV) *Les spécifications des modules pourraient être plus rigoureuses* qu'elles ne le sont avec les CONDITIONS des SUITES d'un élément exprimées par des textes libres.

Le lecteur de ce travail, s'il est informaticien, suggèrera immédiatement que l'on transforme les propositions précédentes en introduisant :

- pour les conditions, des tables de décision, des expressions booléennes ou des règles de grammaire analogues au SI ... ALORS ... SINON des langages de programmation,

- pour le parallélisme d'exécution des éléments d'un module, un formalisme plus rigoureux analogue à ceux utilisés pour représenter des processus comme les systèmes d'exploitation.

Ces propositions d'introduire dans un langage rigoureux certains mécanismes bien connus en programmation et qui s'avèrent utiles dans l'analyse d'un projet ont guidé la version de MACSI-2 présentée dans [L22]. Mais pour être manipulés correctement, ils nécessitent que les personnes qui s'en servent aient une bonne formation en algorithmique.

Or, dans MACSI-1, un des objectifs est d'obtenir que les personnes responsables de la spécification générale de l'application ne soient pas des informaticiens, mais des spécialistes de gestion capables de fournir, après une formation technique très limitée en informatique, la structure générale du projet qu'ils demandent.

Aussi, le niveau de complexité de conception de MODULE proposé dans MACSI-1, au niveau des conditions d'enchaînement des éléments, nous a-t-il semblé le maximum de ce qui peut être demandé à quelqu'un n'ayant pas un profil d'informaticien.

V) *Un élément dans un module représente en fait non pas une action ou une transaction, mais l'exécution d'une action ou d'une transaction.*

C'est pourquoi le schéma des boucles par la règle 6 est possible en exprimant plusieurs activations d'un même processus.

Ceci explique aussi pourquoi l'opération correspondant à une action ou une transaction n'est pas associée de façon biunivoque à un groupe d'exécution et un seul dans l'ensemble du projet. Pouvant avoir plusieurs exécutions dans des contextes différents, c'est à chaque exécution de l'opération que l'on associe un groupe d'exécution et non pas à l'opération elle-même.

Ainsi, dans l'exemple, la transaction T1 est exécutée :

a) automatiquement après le module M4 dans le cadre du module M1,
b) à l'initiative du groupe d'exécution SEC (Secrétariat des inscriptions) dans le cadre du module M4.

B4 - LANGAGE DE DESCRIPTION DE MODULES

La représentation de la description d'un module sous forme de table est difficile à utiliser : cf. les règles 5, 6, 7 et 8. Ces difficultés nous amènent à proposer un langage de description très proche d'une représentation graphique des enchaînements d'actions, de transactions et de modules.

B4-1 - La grammaire

R0 <définition de module> → defmod <code module><libellé><liste de composants>
fmod

R1 <liste de composants> → <composant>[<composant>]*

R2 <composant> → <composant simple> |
<composant complexe>

R3 <composant simple> → a <code action><libellé> par <groupe> |
t <code transaction><libellé> par <groupe> |
m <code module><libellé>

R4 <composant complexe> → si <condition> alors <liste de composants>
[sinon <liste de composants>] fsi |
tantque <condition> faire <liste de composants>
refaire |
indep <liste de composants> findep

R5 <groupe> → <code groupe><libellé> |
auto

R6 <condition> → <texte>

R7 <libellé> → <texte> |
∅

R8 <texte> → * chaîne de caractères *

Remarques :

On peut établir des correspondances entre les règles de cette grammaire et les règles exposées § B-2-2 et § B-2-4, comme par exemple : entre R1, règle 1 et entre R4, règle 2 et règle 3.

B4-2 - Exemples d'application

Reprenons l'exemple du § B-2-3 sur la gestion d'un congrès pour décrire sa structure modulaire avec le langage proposé.

```
defmod   M1 * gestion des inscriptions à un congrès *
    |    m M2 * détermination du programme scientifique *
    |    m M3 * appel aux communications un an à l'avance *
    |    m M20 * édition de la documentation *
    |    a A5 * envoi des bulletins d'inscription *
    |         par TECH * groupe chargé de l'organisation matérielle du congrès
    |    m M4 * procédure périodique d'enregistrement des inscriptions *
    |    m M5 * ouverture du congrès, régularisation des paiements *
    |    m M101 * édition des listes de participants à la demande du res-
    |              ponsable *
    |    m M9 * bilan post-congrès, un mois après la fin du congrès *
fmod
```

Les modules M4, M5, M101, M9 dont l'exécution doit satisfaire des conditions (temporelles pour M4, M5 et M9, de décision pour M101) devront faire l'objet de descriptions contenant ces conditions ; mais on aurait pu tout aussi bien poser ces conditions dans la description du module M1 : par exemple, au niveau du module M101, on aurait pu écrire

```
si * le responsable le décide *
   alors m M101 fsi
```

mais la première description semble meilleure méthodologiquement : décrire l'application comme étant d'abord un ensemble de modules fonctionnels que l'on détaille ensuite.

```
defmod M20
|     a A1 * fixation définitive du programme *
|          par RESPON * groupe de personnes responsables de l'organisation
|                       générale du congrès *
|     indep
|     |     a A2 * édition du programme *
|     |          par TECH
|     |     m M34 * dessin et impression de la fiche d'inscription *
|     findep
fmod
```

Les actions A3 (dessin) et A4 (impression) sont regroupées dans le module M34 de façon à pouvoir être globalement un élément dans un déroulement en parallèle avec l'action A2.

```
defmod M34
|     a A3 * dessin pour l'imprimeur de la fiche d'inscription *
|         par TECH
|     a A4 * impression de la fiche d'inscription *
|         par IMP * cellule chargée de l'impression de la documentation
|                   publicitaire du congrès *
fmod
```

```
defmod M4
|     tantque * la date prévue d'ouverture du congrès est à plus d'une semaine
|         faire m M41 * réception des inscriptions *
|         |     a A10 * envoi avis d'hébergement et de remboursement *
|         |       par TECH
|         |     m M100 * édition des listes provisoires *
|         refaire
fmod
```

```
defmod M41
|     a A6 * réception des bulletins d'inscription remplis *
|          par SEC * secrétariat chargé du suivi des inscriptions *
|     indep
|     |     m M14 * mise à jour du fichier des inscrits *
|     |     a A8 * réservation de l'hébergement demandé *
|     |       par TECH
|     findep
fmod
```

```
defmod M14
|     a A7 * perforation des bulletins *
|          par PERF * groupe de perforation *
|     m M140 * mise à jour du fichier *
fmod

defmod M140
|     t T3 * mise à jour du fichier des inscrits *
|          par TECH
|     t T4 * édition d'un message d'anomalies *
|          par auto
|     t T5 * édition d'une demande de paiement du solde *
|          par auto
|     t T6 * édition de la situation de paiement *
|          par auto
|     tantque * des erreurs d'inscription sont constatées *
|          faire a A7 par PERF
|          |     t T3 par TECH
|          |     t T4 par auto
|          |     t T5 par auto
|          |     t T6 par auto
|          refaire
fmod

defmod M100
|     t T1 * édition des listes des participants inscrits *
|       par auto
fmod

defmod M101
|     si * le responsable le décide *
|        alors
|        |     t T1
|        |       par auto
|        fsi
fmod

... etc ...
```

ANNEXE C

RÉALISATIONS TECHNIQUES :

LE PROCESSEUR DE TRANSACTIONS "NOYAU"

C1 - INTRODUCTION

1 - Historique : NOYAU dénote une famille de sous-programmes Socrate développés à l'IFC* pour répondre aux besoins apparus lors de la réalisation de l'application MEDOC (cf.§C-12-4).

2 - Objectifs : NOYAU a été réalisé pour permettre de construire des prototypes d'applications informatiques en gestion, en améliorant :

- la productivité en programmation,
- la flexibilité de la construction,
- l'autodocumentation de l'application.

Réaliser un prototype d'une application informatique permet :

1) aux différents concepteurs de définir précisément les fonctions qu'ils veulent voir réalisées dans la version opérationnelle ; les qualités d'un tel prototype doivent alors être sa rapidité d'écriture et sa souplesse de modifications.

2) aux utilisateurs de tester le prototype afin de se former à leurs futures responsabilités dans la bonne marche de l'application ; le prototype doit alors être un instrument pédagogique pour la formation d'un personnel compétent.

Lorsqu'une cellule informatique a pour vocation de réaliser de nombreux prototypes pour des applications diverses, elle se dote alors d'outils logiciels lui permettant de faire rapidement face à la demande.

NOYAU est un ensemble de tels outils, pour la construction de prototypes d'applications informatiques.

3 - Choix techniques : Les outils composant NOYAU ont été réalisés en utilisant le Système général de Gestion de Base de Données SOCRATE (IRIS 45). Ils ne sauraient donc être utilisés pour des prototypes non réalisés en SOCRATE.

---

* *Institut de Formation et de Conseil - IUT-B - Place Doyen Gosse - 38031 - GRENOBLE Cedex.*

4 - Principales fonctions :

Ces sous-programmes peuvent être regroupés en trois catégories définissant les fonctions principales de l'outil NOYAU :

* fonction de validation de transactions, réalisée essentiellement par un processeur d'automates d'états finis (cf. § C-3 à C-6)
* fonction de gestion d'un dictionnaire documentaire centralisé dans lequel sont stockés tous les objets gérés dans un prototype ; chaque objet étant défini par un code (qui doit être unique) et divers textes explicatifs (cf. § C-7)
* fonction de définition standard de dossiers de sortie pour restituer les informations gérées par les deux fonctions précédentes (cf. § C-9).

C2 - UNE METHODE UNIFORMISEE DE SAISIE DES DONNEES : LA TRANSACTION

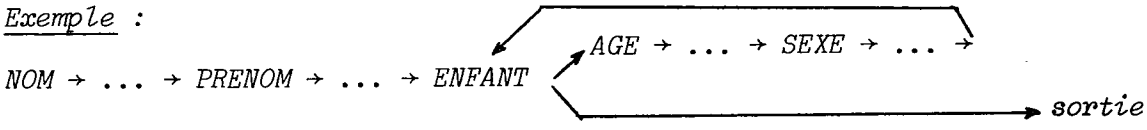
1 - Le schéma de transaction :

Deux approches sont possibles quant à la saisie des données : l'approche classique, en traitement différé (batch) et l'approche conversationnelle.

A chacune de ces approches correspond en général un processus de saisie : document zoné pour le batch, protocole pour le conversationnel ; dans une optique de prototype, il est toutefois souhaitable, au point de vue de la rapidité d'écriture, d'avoir un processus unique (n'ayant alors qu'un seul programme de saisie à écrire).

Un document zoné et un protocole ont quelque chose en commun que nous appelons un SCHEMA de TRANSACTION et qui peut être représenté par un graphe.

Exemple :



2 - La transaction et sa validation :

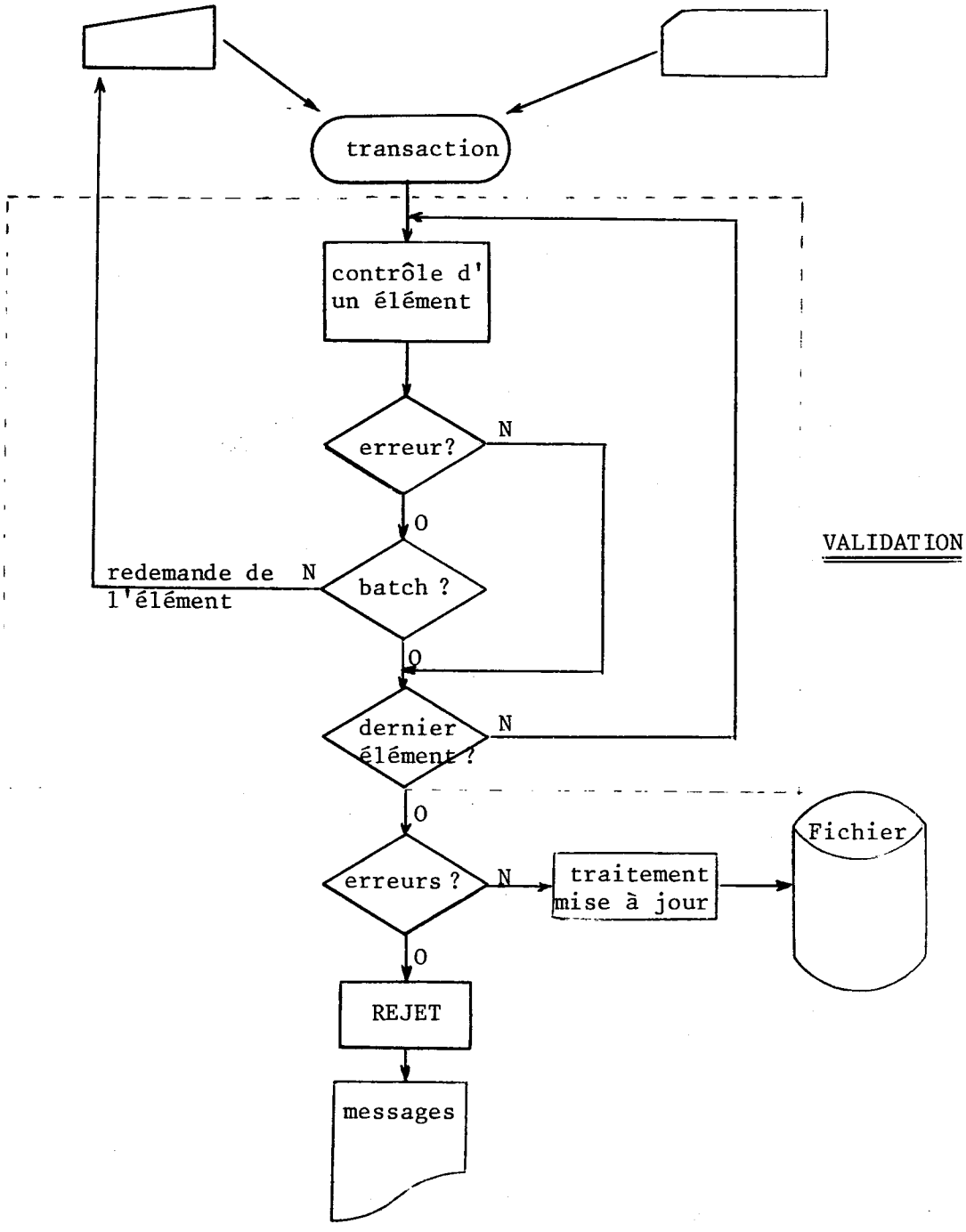
On peut alors envisager que l'utilisateur, en batch ou en conversationnel, transmette ses données par des phrases qui respectent le schéma précédent (les pointillés sont remplacés par des valeurs) :

NOM dupont PRENOM jean ENFANT AGE 2 SEXE masculin
ENFANT AGE 4 SEXE féminin

Une telle phrase est appelée TRANSACTION.

Un schéma de transaction doit être un automate déterministe d'états finis dont les noeuds du graphe sont appelés ELEMENTS (ex. NOM, PRENOM, ...). Une transaction est validée lorsque chaque élément a été contrôlé avec succès (ex. NOM dupont, PRENOM jean, ...).

La validation d'une transaction peut être décrite par le processus suivant :



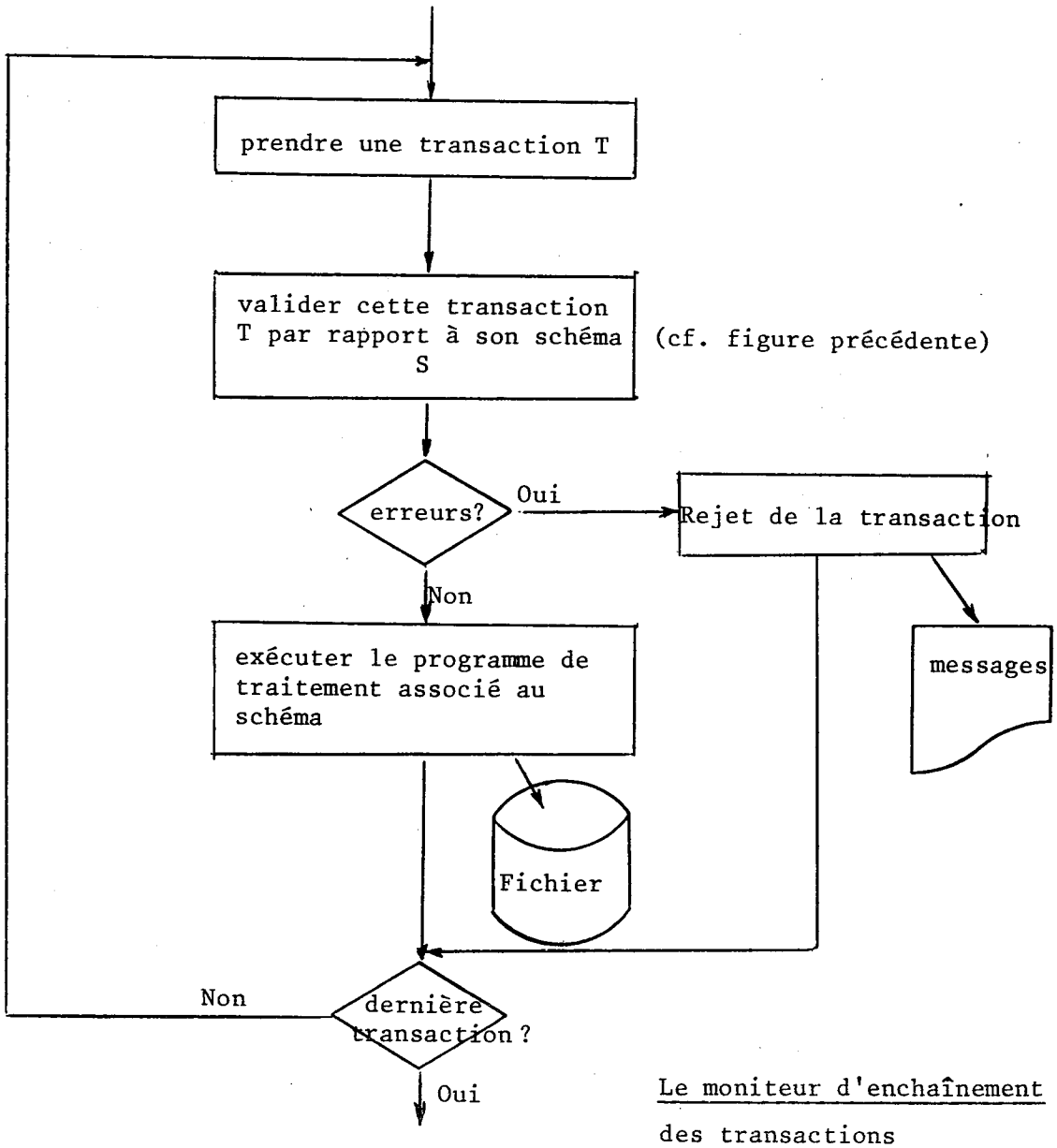
Validation d'une transaction

3 - Le moniteur d'enchaînement des transactions :

Nous venons d'étudier l'outil permettant de valider une transaction ; lorsque le dernier mot d'une transaction est validé, la phase de traitement peut alors être effectuée.

A chaque schéma possible correspond une procédure de traitement (cf.§ C.3.4) (dont l'écriture est évidemment à la charge du concepteur du prototype) ; afin de respecter la souplesse du prototype, il est nécessaire que ces procédures soient indépendantes entre elles, et qu'un certain mécanisme soit capable, une transaction étant donnée,de l'aiguiller vers la procédure de traitement correspondante.

Le moniteur d'enchaînement des transactions réalise alors le processus suivant :



Le moniteur d'enchaînement des transactions

C3 - DEFINITION D'UN SCHEMA DE TRANSACTION

1 - Un outil de validation d'un schéma : le schéma des schémas

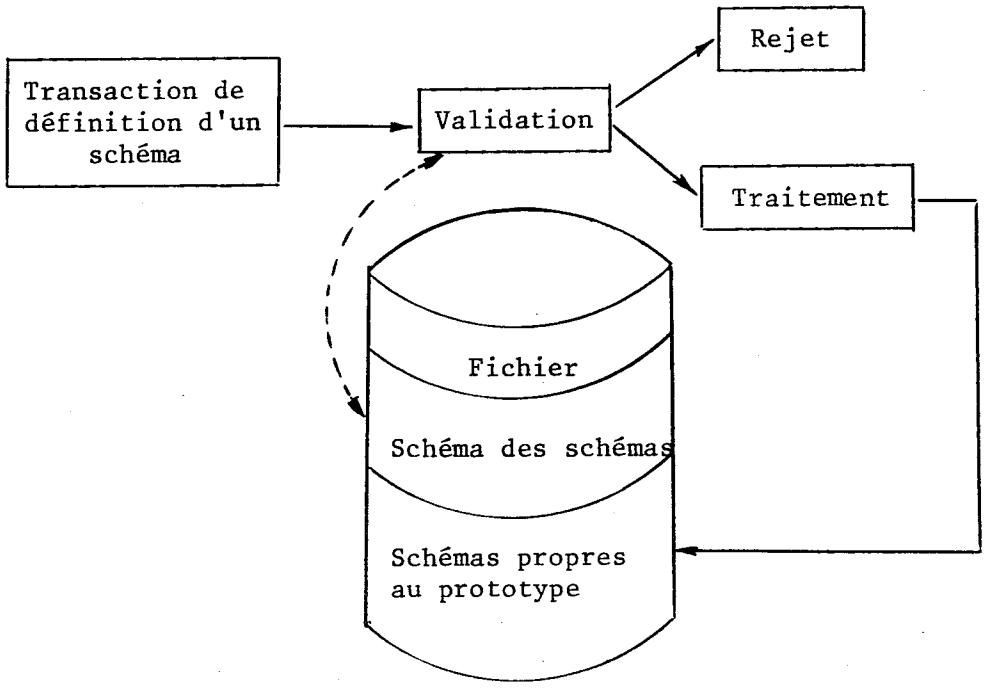
Lorsque plusieurs schémas de transaction sont possibles, la procédure de validation d'une transaction quelconque se décompose généralement en plusieurs sous-procédures : une par schéma possible.

Nous pouvons remarquer que la définition d'un schéma de transaction spécifique à un prototype est aussi une transaction pour NOYAU : c'est la définition d'un automate d'états finis. Ainsi, dans NOYAU, on définit un *schéma général de transaction* permettant de spécifier d'autres schémas.

Dans une telle approche, l'écriture d'une procédure de validation pour un schéma spécifique est donc remplacée par la saisie des données définissant ce schéma (d'où un gain de temps et une plus grande souplesse).

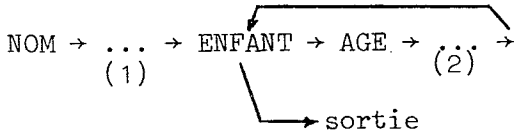
On dispose donc ici d'une seule procédure (algorithme de cheminement dans un graphe quelconque), les différents graphes possibles étant décrits dans la base.

Dans tout prototype utilisant un tel outil, il existera donc, dès la phase d'initialisation, un schéma général de transactions, celui qui permet de décrire les schémas propres au prototype ; nous l'appelons le *SCHEMA des SCHEMAS*.



2 - Contrôles sémantiques

Saisir un schéma de transaction sous forme de données (c'est-à-dire saisir le graphe associé) ne permet pas toutefois de définir complètement la procédure de validation associée à ce schéma ; certains contrôles, en effet, ne sont pas exprimables en terme de cheminement dans un graphe, mais demandent l'écriture de programmes spécifiques, par exemple :



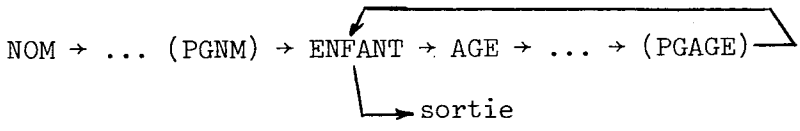
(1) vérifier que la chaîne alphanumérique saisie correspond au NOM d'une PERSONNE déjà rentrée dans le fichier.

(2) vérifier que la valeur saisie est numérique et comprise entre 1 et 100.

L'écriture des programmes effectuant ces contrôles spécifiques reste évidemment à la charge du concepteur du prototype. Il faut l'ensemble de ces programmes et le graphe associé au schéma de transaction, pour réaliser complètement la procédure de validation correspondante.

Nous avons retenu la solution suivante :

(1) Lorsque le graphe associé est saisi, est saisi également pour tout noeud sur lequel doit être effectué un contrôle, un nom définissant ce contrôle.



(2) Les programmes réalisant ces contrôles sont écrits en respectant certaines conventions (cf. § C-4) :

procédure : PGAGE

(3) La procédure générale de validation d'un schéma quelconque est alors capable pour chaque schéma particulier et à chaque noeud de ce schéma :

- de savoir si un contrôle existe,
- s'il existe, de savoir son nom et de pouvoir alors appeler le programme correspondant.

Ainsi sera réalisée de façon complète la procédure de validation d'un schéma de transaction.

3 - Principes généraux de définition d'un schéma

Appelons "règle" une partie d'un schéma.

(1) Soit la règle S :

A → ... → B

où A et B sont des mots-clés et où ... est à remplacer par une valeur, cette valeur devant être numérique.

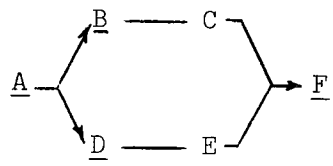
Si nous donnons le nom C au noeud du graphe noté ..., alors la règle S peut être décrite par :

```
mreg S
  | clé A
  | num C
  | clé B
freg
```

Un tel graphe sera désormais écrit sous la forme :

A → C → B, où les mots-clés sont soulignés.

(2) Soit maintenant la règle T :



Pour décrire cette règle, une sous-règle R sera introduite, permettant de décrire les deux alternatives :

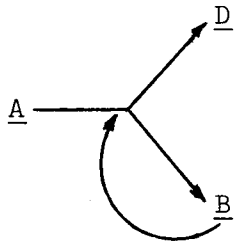
B → C et D → E

Ainsi la règle T pourra être décrite par les trois règles suivantes :

```
mreg T          mreg R          mreg R
  | clé A         | clé B         | clé D
  | reg R         | num C         | num E
  | clé F        freg            freg
freg
```

La règle "MREG ... FREG" apparaît ainsi comme donnant la possibilité de décrire une nouvelle alternative à une règle donnée.

(3) Soit la règle P :



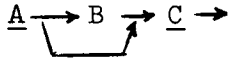
selon laquelle seraient correctes les phrases suivantes :

A D
A B D
A B B D
A B B B D
etc ...

Nous la décrirons par les trois règles successives :

```
mreg P          mreg Q          mreg Q
  | clé A         | clé D         | clé B
  | reg Q        freg            | goto Q
freg                            freg
```

(4) Soit enfin la règle U :



qui présente une alternative vide et qui sera décrite par les règles :

```
mreg U
  | clé A
  | reg V
  | clé C
freg
```

```
mreg V
  | num B
freg

mreg V
  | null
freg
```

ou

```
mreg V
  | num B
ffreg
```

(5) Nous avons vu enfin que sur certains noeuds du graphe pouvaient exister des contrôles sémantiques auxquels il est nécessaire de donner un nom, par exemple :

AGE → VAL (PGAGE) → FAGE

Un tel schéma W pourra être décrit par la règle :

```
mreg W
  | clé AGE
  | num VAL semant PGAGE
  | clé FAGE
freg
```

et par un programme SOCRATE respectant certaines conventions :

```
proc : PGAGE
     {
   /* vérifie qu'une valeur numérique */
   /* est comprise entre 1 et 100 */
     }
```

4 - Notations retenues

La notation à laquelle nous venons d'aboutir est à la base de celle que nous utilisons pour définir de nouveaux schémas de transaction ; nous la complétons dans un souci de documentation ; ainsi, le commentaire apparaissant dans l'exemple précédent au niveau du programme PGAGE sera directement "placé" dans la règle :

```
mreg W
  | clé AGE
  | num VAL semant PGAGE * cette valeur doit être comprise
  | entre 1 et 100 *
  | clé FAGE
freg
```

Un tel commentaire a ici pour but de définir fonctionnellement le programme PGAGE.

Les conventions suivantes de documentation seront retenues :

(1) Lorsqu'un nom de programme sémantique est mentionné pour la première fois, un commentaire (texte libre encadré par deux étoiles) devra lui être associé.

(2) Lorsqu'un nom de noeud de graphe est mentionné pour la première fois, un libellé (chaîne alphanumérique d'au plus 60 caractères, encadrée par deux étoiles) devra lui être associé , ainsi qu'éventuellement un commentaire précédé du mot-clé DEF.

Exemple :

```
mreg PAYROLL
   | clé NOM * permet d'introduire le nom d'une personne *
   | code NOMPERS * nom d'une personne *
   |   def * le nom d'une personne est un code, c'est-à-dire qu'il ne
   |         peut exister dans le fichier d'autres objets ayant ce même
   |         code *
   |   semant PGNM * ce code doit être celui d'une personne dans le
   |                 fichier ; cette personne doit être de sexe masculin *
   :   - - - - - -
```

freg, correspond au graphe :

NOM → *NOMPERS (PGNM)* → - - - -

Autres conventions :

(1) La règle MREG ... FREG permet, nous l'avons vu, de définir une nouvelle alternative à une règle initialisée. Une règle est initialisée si elle a été précédemment déclarée à l'aide du mot-clé REG, par exemple :

```
mreg U          <=>   a) définition d'une nouvelle alternative
   | clé A                à la règle U
   | reg V             b) initialisation d'une nouvelle règle V.
   | clé C
freg
```

Il existe toutefois une règle de plus haut niveau, dénommée ANAL ; elle est prédéclarée dans le système ; définir un nouveau schéma de transaction consiste à décrire une nouvelle alternative à cette règle.

Définir un nouveau schéma de transaction nécessite d'autre part (cf.§C-2.3) de préciser le nom du programme de traitement qui permettra alors au "moniteur d'enchaînement" de l'appeler après la validation d'une transaction correspondant à ce schéma.

Pour définir un nouveau schéma, nous utiliserons donc une transaction du type :

```
mreg ANAL
  }
  }
freg exec PROG-TRAITEMENT
```

(où PROG-TRAITEMENT est le nom du programme qui traitera les transactions réglées par ce nouveau schéma que l'on se définit).

(2) Les noeuds de graphe sont de sept types possibles :

- mot-clé,
- numérique,
- code,
- mot,
- ligne,
- texte,

ou - liste de valeurs numériques.

Ces différents types sont définis dans le paragraphe suivant. Précisons, dès maintenant, qu'un code est une chaîne alphanumérique d'au plus 12 caractères ; tout code défini au cours d'une transaction sera placé dans le dictionnaire centralisé (cf.§C-7) avec son libellé et éventuellement ses textes explicatifs ; deux objets différents du fichier ne peuvent avoir le même code.

5 - Représentation graphique du schéma des schémas

Ce schéma sera présent dans toute application utilisant NOYAU ; c'est celui qui règle toute transaction ayant pour but de décrire les différents schémas propres à l'application.

Afin de faciliter la compréhension de ce graphe, nous adopterons les conventions suivantes :

(1) les noeuds de type mot-clé seront écrits en minuscules et soulignés, les autres en majuscules,

(2) A ⟶ B indique que le noeud B suit le noeud A.

Remarque : Signification du GOTO

Supposons les transactions successives :

(1) mreg R1
clé A
reg R2
clé B
freg

(2) mreg R2
clé C
freg

(3) mreg R2
clé D
freg

(4) mreg R3
clé E
goto R2
freg

Elles traduisent le graphe suivant :

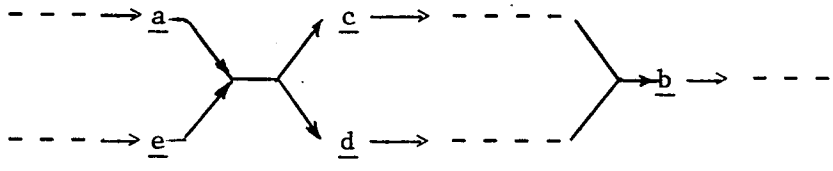
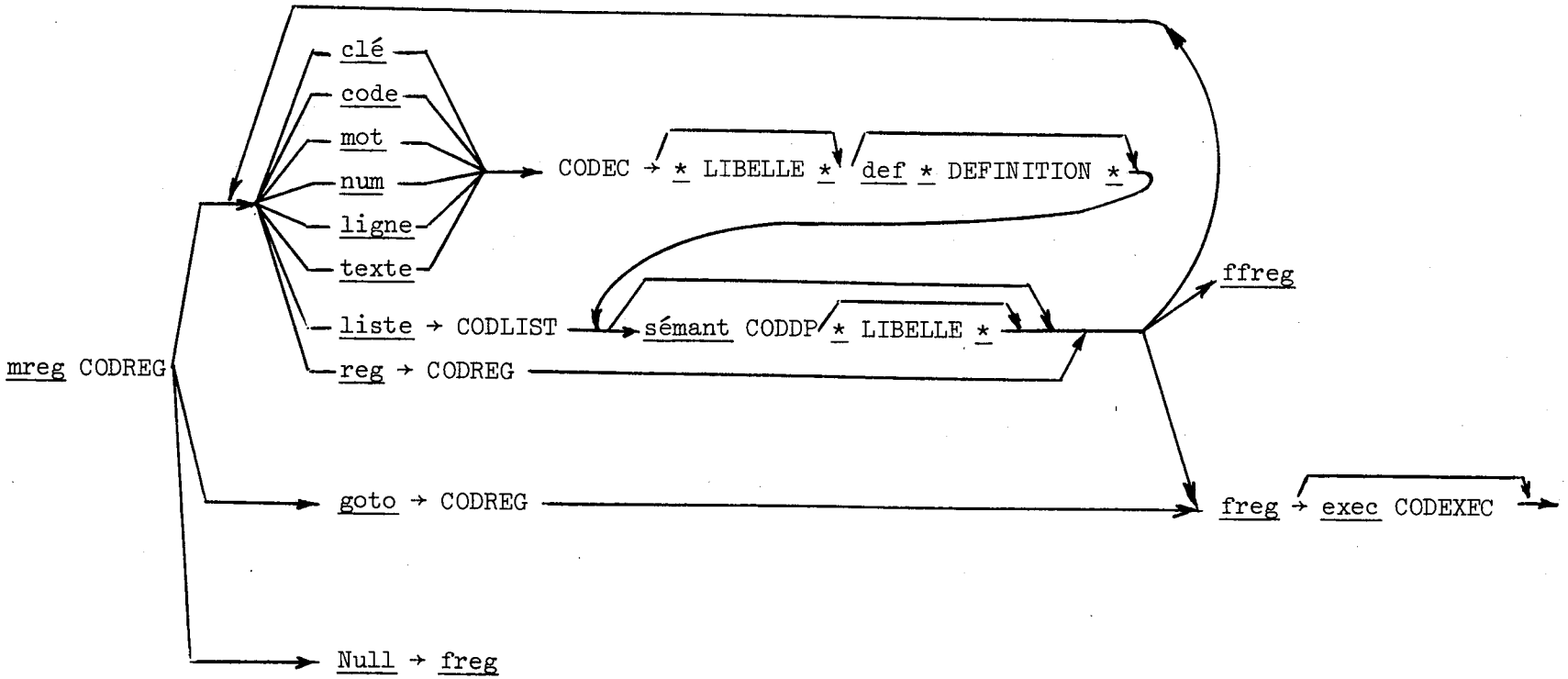


Schéma des schémas :



Remarque :

La première alternative décrite d'une règle ne doit pas commencer par REG, GOTO ou NULL ; cette contrainte est définie dans un programme de contrôle associé à ce schéma (cf. les pages suivantes).

6 - Auto-description du schéma des schémas

Ce schéma des schémas permet de décrire d'autres schémas ; décrit avec sa terminologie propre, on aurait les règles successives suivantes :

```
mreg ANAL
   | clé MREG * permet de définir une nouvelle alternative à une règle *
   |     def * la définition d'un nouveau schéma de transaction se fait en
   |           définissant une nouvelle alternative à la règle ANAL qui est
   |           prédéfinie dans le système *
   | code CODREG * code de règle *
   |     semant S-CODREG * ce code doit être celui de la règle pour
   |            laquelle on veut définir une nouvelle alternative ; cette
   |            règle doit donc avoir déjà été définie. Une nouvelle
   |            alternative ne peut toutefois être décrite si la précé-
   |            cente ne commençait pas par un élément de type CLE, MOT,
   |            LIGNE ou TEXTE *
   | reg ENTRE
freg exec E-MREG

mreg ENTRE
   | clé NULL * spécifie une alternative vide *
   |     def * une alternative vide pour une sous-règle permet de spéci-
   |           fier que cette sous-règle est facultative ; on pourra
   |           éviter d'utiliser une transaction spécifiant cette alter-
   |           native vide, en terminant la transaction au cours de
   |           laquelle a été spécifiée la précédente alternative de cette
   |           sous-règle, par le mot-clé FFREG, au lieu de FREG *
   |     sémant S-NULREGOTO * la première alternative décrite d'une règle
   |           ne peut être une alternative vide ; de même, elle ne peut
   |           être une alternative dont la description commencerait par
   |           un branchement : GOTO, ou par une initialisation de nouvelle
   |           règle : REG *
   | clé FREG * fin de spécification d'une alternative de règle *
freg

mreg ENTRE
   | reg NONNULL
freg
```

```
mreg NONNULL
   | clé GOTO * permet de spécifier un branchement *
   |      def * un branchement sur une règle permet de spécifier qu'en ce
   |             point de l'analyse, toutes les alternatives de la règle
   |             sur laquelle on se branche (toutes les alternatives déjà
   |             définies ou qui seront définies ultérieurement) seront
   |             autorisées *
   |          semant S-NULREGOTO
   |     code CODREG
   |          semant S-GOTOREG * la règle sur laquelle on se branche doit avoir
   |                 précédemment été définie *
   |     clé FREG
   |          semant S-FREG * la fin de spécification d'une alternative à la
   |                 règle ANAL doit être suivie du mot-clé EXEC qui permettra
   |                 d'introduire le nom du programme de traitement associé au
   |                 schéma de transaction que l'on commence à décrire *
   |          reg EXEC
freg

mreg EXEC
   | clé EXEC * introduit le nom du programme de traitement d'un schéma de
   |            transaction *
   | code CODEXEC * nom du programme de traitement *
ffreg

mreg NONNULL
   | reg CAS
   | reg SORT
freg

mreg SORT
   | clé FREG
   |     semant S-FREG
   | goto EXEC
freg
```

```
mreg SORT
   | clé FFREG * fin de spécification d'une alternative de règle et création
   |             d'une alternative vide *
freg

mreg SORT
   | goto NONNULL
freg

mreg CAS
   | clé REG * permet d'initialiser une nouvelle règle *
   |     def * une nouvelle règle sera initialisée en tout point du graphe
   |           d'où arrivent ou d'où repartent plus d'une flèche *
   |     semant S-NULREGOTO
   | code CODREG
   |     semant S-REGCOD * ce code ne peut pas être celui d'une règle déjà
   |           définie *
freg

mreg CAS
   | clé LISTE * introduit un noeud de type liste de valeurs numériques *
   |     def * en un tel noeud d'une transaction, une valeur numérique
   |           devra être donnée à choisir parmi une certaine liste de
   |           valeurs possibles *
   | code CODELIST * code de liste de valeurs *
   |      semant S-CODELIST * ce code doit être celui d'une liste précédemment
   |           décrite à l'aide du schéma de transaction CRELIST *
   | reg.SEMANT
freg

mreg CAS
   | reg TYPE
   | code CODEC * code d'un noeud du graphe *
   |     semant S-CODE * si ce code est nommé pour la première fois, alors
   |            son libellé doit être immédiatement spécifié *
   | reg LIBDEF
   | goto SEMANT
freg
```

```
mreg LIBDEF
   | ligne LIBELLE * libellé d'un élément *
   | reg DEFINIT
ffreg

mreg DEFINIT
   | clé DEF * introduit un texte de définition *
   |    def * un texte de définition ne pourra être spécifié que lorsque
   |           l'élément à définir est nommé pour la première fois. Toute-
   |           fois, des commentaires pourront être ultérieurement attachés
   |           à l'élément au cours de transactions réglées par le schéma
   |           CREEX *
   | texte TEXTDEF * texte de définition *
ffreg

mreg SEMANT
   | clé SEMANT * permet de spécifier un contrôle *
   |    def * un contrôle est rattaché à un noeud du graphe lorsqu'un pro-
   |           gramme spécifique est à activer en ce noeud *
   | code CODOP * code d'un contrôle *
   |    def * un code de contrôle peut être différent du nom du programme
   |           qui réalise ce contrôle ; un appel associatif devra donc être
   |           spécifié par le concepteur, pour mettre en correspondance ce
   |           code et ce nom *
   |    semant S-CODOP * si ce code est nommé pour la première fois, il
   |           devra être immédiatement suivi d'un texte précisant la nature
   |           du contrôle correspondant *
   | reg TEXTSEM
ffreg

mreg TEXTSEM
   | texte COMMENT-OP * définition fonctionnelle d'un contrôle sémantique *
ffreg
```

```
mreg TYPE
   | clé CODE * introduit un élément de type code *
   |    def * un code est une chaîne alphanumérique d'au plus 12 caractères,
   |          sans blancs intermédiaires *
freg

mreg TYPE
   | clé MOT * introduit un élément de type mot *
   |    def * un mot est une chaîne alphanumérique d'au plus 30 caractères,
   |          encadrée par deux étoiles *
freg

mreg TYPE
   | clé LIGNE * introduit un élément de type ligne *
   |    def * une ligne est une chaîne alphanumérique d'au plus 60 caractères,
   |          encadrée par deux étoiles *
freg

mreg TYPE
   | clé TEXTE * introduit un élément de type texte *
   |    def * un texte est limité à 200 lignes de 60 caractères au plus
   |          chacune, et encadré par deux étoiles *
freg

mreg TYPE
   | clé NUM * introduit un élément de type numérique *
freg

mreg TYPE
   | clé CLE * introduit un élément de type mot-clé *
freg
```

Les programmes de contrôles sémantiques (sémant) seront décrits en C-4-8.

7 - Exemples de définition de schémas

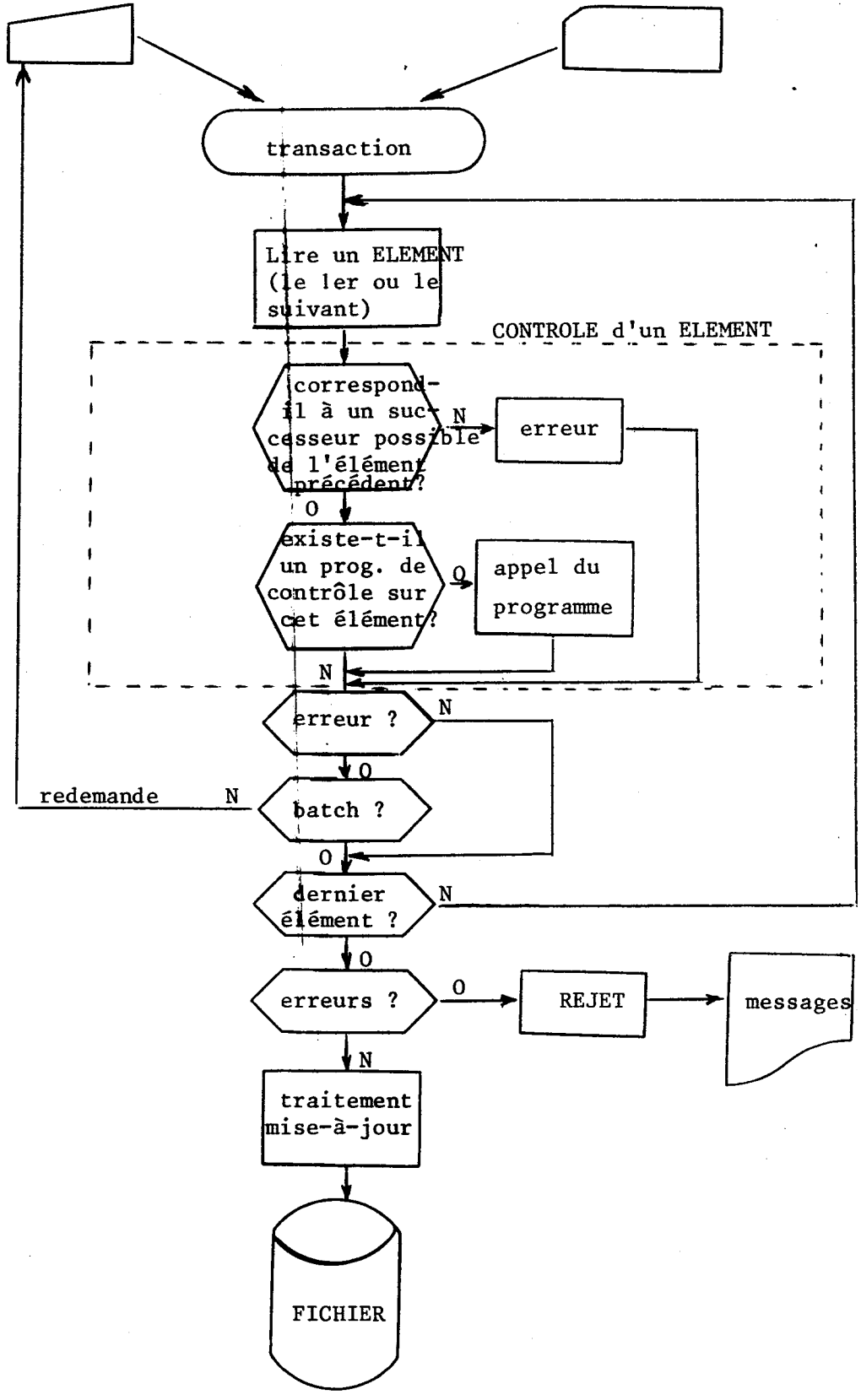
Nous venons de définir le schéma permettant de décrire d'autres schémas. Dans le paragraphe C-8, nous donnons des exemples de telles descriptions : les schémas CREME, CRELIST et CREEX qui sont définis dans NOYAU et donc dans toute application construite avec NOYAU ; ils permettent de décrire ultérieurement des messages d'erreurs, des listes de valeurs et des textes explicatifs (cf. § C-8).

C4 - ECRITURE DES PROGRAMMES DE CONTROLE

1. Validation d'une transaction

Nous avons donné le processus général de validation d'une transaction (cf.§ C-2.2) et introduit les contrôles sémantiques (cf.§ C-3.2) nous pouvons maintenant décrire avec plus de précision ce processus, notamment la partie "contrôle d'un élément" (contrôle du nom de l'élément dans le cas d'un noeud du type mot-clé, sinon contrôle de la valeur de l'élément pour les autres types).

Les programmes de contrôles sémantiques étant appelés par NOYAU en cours de validation d'une transaction, il est important que le concepteur du prototype respecte des contraintes précises dans leurs écritures, relatives à l'interface avec NOYAU.


transaction
Lire un ELEMENT (le 1er ou le suivant)
CONTROLE d'un ELEMENT
correspond-il à un successeur possible de l'élément précédent?
N
erreur
O
existe-t-il un prog. de contrôle sur cet élément?
O
appel du programme
N
erreur ?
N
O
redemande
N
batch ?
O
dernier élément ?
N
O
erreurs ?
O
REJET
messages
N
traitement mise-à-jour
FICHIER

2 - Rôle des variables Z1 et Y1

Avant l'appel d'un programme de contrôle, NOYAU range dans la variable Z1 la chaîne alphanumérique qui vient d'être lue et sur laquelle le contrôle veut être effectué ; si, de plus, cette chaîne corréspond à une valeur numérique, cette valeur sera rangée dans Y1.

	Variables d'entrée d'un programme de contrôle
Z1	contient la valeur de l'élément courant de la transaction
Y1	=Z1 si la valeur de l'élément courant doit être numérique

3 - Contrôle sur la valeur d'un élément

Exemple :

- *Soit dans un schéma de transaction, la séquence*

  *... num AGE semant PGAGE * cette valeur doit être comprise entre 1 et 100 *
  ...*

- *On pourra alors implémenter le contrôle PGAGE à l'aide du programme*

```
:defpro PGAGE :exp
| si Y1 < 1 ou Y1 > 100 alors ERREUR (102)
| fin
:fdef
```

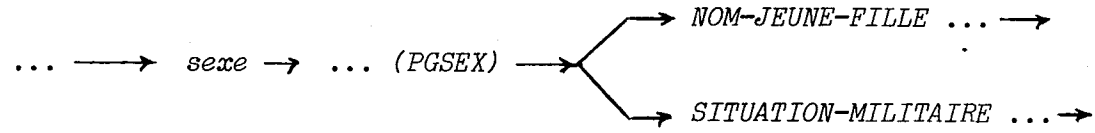
(où ERREUR est une macro utilitaire qui réalise l'édition d'un message d'erreurs repéré par son numéro, cf. § C-4-7).

4 - Contrôle sur un choix de parcours

(1) Nous avons vu qu'une transaction correspondait à un certain cheminement dans un graphe donné ; un programme de contrôle peut définir des restrictions sur l'ensemble des successeurs possibles du noeud auquel il est rattaché.

Exemple :

- *Soit dans un schéma de transaction, la séquence :*



PGSEX : si la personne a pour sexe la valeur : FEMININ, alors on devra définir son nom de jeune fille, sinon on définira sa situation militaire.

- *Les transactions suivantes seraient alors correctes :*
  ... féminin NOM-JEUNE-FILLE ...
  ... masculin SITUATION-MILITAIRE ...

tandis que seraient incorrectes :
  ... féminin SITUATION-MILITAIRE ...
  ... masculin NOM-JEUNE-FILLE ...

(2) Conventions et rôles des variables Z2, Y14 et Y23

- En entrée d'un programme de contrôle, les variables Z2 et Y14 sont mises à "indéfinie" par NOYAU.
- Si en sortie, Z2 a une valeur, alors le mot suivant dans la transaction est lu et comparé à Z2 ; si Y14 est toujours indéfini, NOYAU contrôle alors qu'ils sont égaux ; si Y14 n'est plus indéfini, NOYAU contrôle qu'ils sont différents.
  En cas d'erreur au contrôle, NOYAU exécutera une instruction : ERREUR(Y23).

Variables de sortie d'un programme de contrôle		
Z2 = U	∀ Y14 ∀ Y23	pas de restrictions sur la valeur de l'élément suivant
Z2 ≠ U	Y14 = U Y23 ≠ U	la valeur de l'élément suivant doit être égale à Z2 sinon ERREUR (Y23)
Z2 ≠ U	Y14 ≠ U Y23 ≠ U	la valeur de l'élément suivant doit être différente de Z2, sinon ERREUR (Y23)

(3) *Exemple* :

Le contrôle PGSEX précédent peut alors être réalisé par un programme tel que :

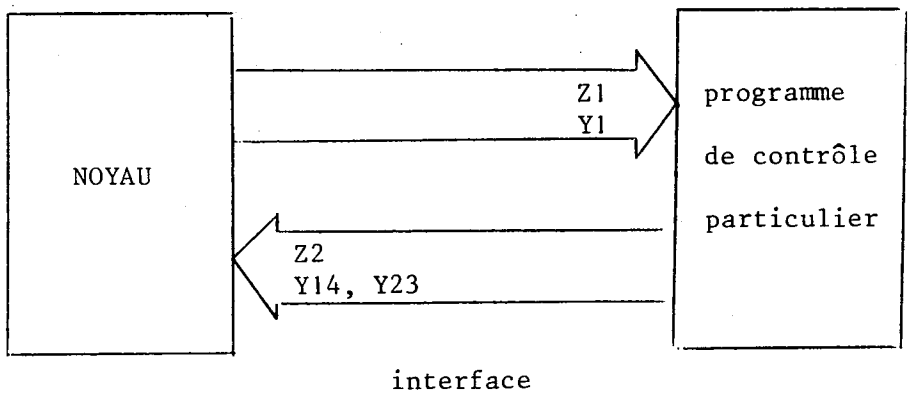
```
:defpro PGSEX
:exp
| m Z2 = 'NOM-JEUNE-FILLE' m Y23 = 105
| si Z1 = 'masculin' alors m Y14 = 1 fin
:fdef
```

5 - Contraintes sur les variables Xi, Yi, Zi

Un va-et-vient s'opérant entre les programmes de contrôle et les programmes du NOYAU, certaines contraintes quant à l'utilisation des variables SOCRATE doivent être respectées :

(1) Les variables de communication ont été définies au cours des deux paragraphes précédents ; il s'agit, rappelons-le, de :

- Z1, Z2
- Y1, Y14, Y23



interface

(2) Sont utilisées par NOYAU, les variables :

- Z1, Z2, Z3, Z5, Z6
- Y1, Y14, Y15, Y16, Y17, Y18, Y19, Y20, Y21, Y22, Y23, Y24 et Y25
- X7, X9.

(3) De ces variables utilisées par NOYAU, les suivantes ne doivent pas être modifiées par un programme de contrôle :

- Z1
- Y15, Y16, Y17, Y18, Y19, Y20, Y21, Y22, Y24, Y25
- X7.

(4) Rôle de X8 : La variable X8 dans un programme de contrôle ne peut être utilisée qu'associée à l'entité DICO (dictionnaire centralisé, cf. § C-7) et définie en contexte du programme, par exemple :

```
:defpro CONTROL
:contxt Xo d X8 = un DICO
:exp
  §
:fdef ?
```

6 - Appel associatif des programmes de contrôle

Un programme de contrôle étant écrit, il est nécessaire d'indiquer à NOYAU que ce programme réalise tel contrôle défini jusque-là fonctionnellement et mentionné sous un certain nom dans un schéma de transaction.

La mise en correspondance d'un nom de contrôle et du nom du programme qui implémente ce contrôle se fait à l'intérieur du programme U-SEMAN qui effectue en fait l'appel associatif des programmes de contrôle.

```
:defpro U-SEMAN  :exp
   faire
      d x8 = un DICO
      si Z3 = 'PGAGE' alors exec PGAGE sortie fin
      si Z3 = 'PGSEX' alors exec PGSEX sortie fin
      si Z3 = 'P3' alors exec PN15 sortie fin
      sinon
         i '/// CONTROLE ABSENT'
      fin
   fin
:fdef
```

Il est naturellement conseillé de donner un nom identique au programme et au contrôle correspondants.

Ce programme U-SEMAN sera donc à modifier et à recompiler dès qu'une nouvelle association devra être introduite.

C'est une contrainte de la version Socrate (V1-5) utilisée, qui ne permet pas un appel associatif standard de programmes.

7 - Erreurs sur un élément

Nous avons vu aux paragraphes précédents que, lors de l'écriture des programmes de contrôle, l'analyste est amené pour chaque contrôle à définir les erreurs associées en utilisant la macro ERREUR.

A ce stade, l'erreur n'est définie que par un numéro :
par exemple : ERREUR (102) dans le programme PGAGE.

A chaque erreur peut être associé un libellé ainsi qu'éventuellement un texte explicatif, qui seront alors délivrés à l'utilisateur lorsque se produira l'erreur ; cette association est décrite, en utilisant la règle de description CREME (voir § C-8-1).

Le message associé à l'ERREUR (102) pourra alors ainsi être fourni à NOYAU :

```
CREME 102 *
   |   valeur non comprise entre 1 et 100 *
FCREME
```

8 - Exemples de programmes de contrôle

(Voir s-creme, s-crelist, s-creex § C-8).

A titre d'exemples, nous pouvons donner les programmes de contrôle du schéma des schémas (cf. § C-3-6).

```
:DEFPRO S-CODREG :CONTXT X0 :EXP
FAIRE
   M X1 = DERNIER DE UNE REGLE X2 AVEC NOM = Z1 ;
   SI PAS X2 ALORS ERREUR (27) SORTIE FIN
   SI PAS X1 ALORS ERREUR (28) SORTIE FIN
   M Y3 = NUMDE X2
   SI PAS SORT DE X2 ALORS M Y2 = 0 SINON M Y2 = 1 FIN
FIN
:FDEF ?

:DEFPRO S-NULREGOTO :CONTXT X0 :EXP
SI Y2 = 0 ALORS ERREUR (29) FIN
:FDEF ?

:DEFPRO S-GOTOREG :CONTXT X0 :EXP
SI PAS UNE REGLE AVEC NOM = Z1 ; ALORS ERREUR (30) FIN
:FDEF

:DEFPRO S-FREG :CONTXT X0 :EXP
M Z2 = 'EXEC' M Y23 = 31
SI Y3 = 1 ALORS M Y14 = 0 FIN
:FDEF ?

:DEFMAC U-SLIBELEM ; :EXP
M Y11 = :1:
EXEC U-INIT
FAIRE
   SI Y12 > 1 ALORS
      ERREUR (53)
      SORTIE
   FIN
   M Z2 = '*'
   M Y23 = 34
   SI Y12 = 0 ALORS M Y14 = 0 FIN
FIN
:FDEF ?

:DEFPRO S-RESCHOD :CONTXT X0 :EXP
M Y2 = 1
SI EXISTE UNE REGLE AVEC NOM = Z1 ; ALORS ERREUR (32) FIN
:FDEF ?

:DEFPRO S-CODE :CONTXT X0 X8 = UN DICO :EXP
U-SLIBELEM 31
```

```
M Y2 = 1
:FDEF

:DEFPRO S-CODOP :CONTXT X2 D X3 = UN DICO :EXP
U-SLIBELEM &
:FDEF

:DEFPRO S-CODELIST :CONTXT X2 D X8 = UN DICO :EXP
M Y11 = 40 EXEC U-INIT
SI Y12 7= 0 ALORS ERREUR (55) FIN
M Y2 = 1
:FDEF?
```

C5 - ECRITURE DES PROGRAMMES DE TRAITEMENT

Nous avons vu comment à chaque schéma de transaction devait correspondre un programme de traitement (cf.§ C-3-4) qui serait appelé par le moniteur d'enchaînement (cf.§ C-2-3) pour toute transaction réglée sur ce schéma et ayant subi avec succès la phase de validation.

Nous décrivons dans ce paragraphe l'interface avec NOYAU et les contraintes à respecter dans l'écriture de ces programmes de traitement.

1 - Lecture de la transaction : la macro-instruction LEC n et la variable Z1

Une transaction est composée de plusieurs mots consécutifs ; lors de l'appel du programme de traitement, le premier mot de la transaction est placé par NOYAU dans la variable Z1 ; les mots suivants pourront être lus par exécution de la macro-instruction LEC :
Z1 contenant un certain mot de la description, l'exécution de LEC n place dans Z1 le $n^{ième}$ mot suivant dans la transaction.

Exemple :

. *avant :* - - - *ENFANT* → *SEXE* → *masculin* → - - -

Z1 = [*ENFANT*]

. *lec 2 :*

. *après :* ---- *ENFANT* → *SEXE* → *masculin* → - - -

Z1 = [*masculin*]

Cas des mots et lignes

Un mot (une ligne) est une chaîne alphanumérique d'au plus 30 (60) caractères, encadrée par deux étoiles.

Lorsque sur exécution de la macro-instruction LEC, Z1 contient une * de début de ligne ou mot, alors l'exécution de LEC 1 placera dans Z1 les 30 premiers caractères, tandis que l'exécution de LEC 3 placerait dans Z1 l'* de fin ; dans le cas d'une ligne, les 30 derniers caractères seraient atteints par exécution de LEC 2.

Cas des textes

Un texte est une chaîne alphanumérique d'au plus 200 lignes de 60 caractères chacune et encadrée par deux étoiles.

On utilisera l'utilitaire U-BGEX et l'entité EXPLIC (cf. § C-7) lorsque sur exécution de la macro-instruction LEC, Z1 contient une * de début de texte, l'exécution du programme U-BGEX permettra alors de lire et de ranger ce texte dans une EXPLIC.

En sortie de U-BGEX, X9 pointera sur l'EXPLIC créée tandis que Z1 contiendra le mot de la transaction qui suit l'étoile de fin.

La chaîne de caractère RAS, pour Rien A Signaler (U pour Undefined), peut également être prise en compte par U-BGEX ; en sortie, Z1 contiendra alors le mot qui suit RAS (ou qui suit U) ; X9 pointera sur la première EXPLIC (ou sera indéfini).

Exemple :

. *avant :* --- *ADRESSE* * ... *un texte* ... * *AGE* ---

Z1 = [*ADRESSE*]

. *lec 1*
exec U-BGEX

. *après :* --- *ADRESSE* * .. *un texte* ... * *AGE* ----

Z1 = [*AGE*]

2 - Contraintes sur les Xi, Yi, Zi

(1) Sont utilisées par le moniteur d'enchaînement des transactions et ne peuvent pas être modifiées par un programme de traitement, les variables :

- Y16, Y18, Y21, Y22, Y24, Y25.

(2) Les variables X8 et X9 ne peuvent être utilisées que dans un certain contexte : DICO et EXPLIC (cf. § C-7). On aura par exemple :

```
:defpro TRAITEMENT
:contxt  X0 d X8 = un DICO d X9 = une EXPLIC
:exp
  }
:fdef
```

(3) Pour la variable Z1 se reporter au § C-5-1.

3 - Appel associatif des programmes de traitement

Nous avons vu que pour tout schéma de transaction, un nom de procédure de traitement devait être spécifié en même temps que le graphe correspondant, et comment ce nom était spécifié (cf. § C-3-4) :

```
mreg ANAL
  }
freg exec PROG-TRAITEMENT
```

Lorsque le programme SOCRATE qui réalise ce traitement est écrit, son nom doit être mis en correspondance avec le nom spécifié lors de la saisie du graphe.

Cette mise en correspondance se fait à l'intérieur du programme U-EXECUTION, d'une manière analogue à celle que nous avons rencontrée pour l'attachement des programmes de contrôle (cf. § C-4-6).

```
:defpro U-EXECUTION :exp
faire
  d X8 = un DICO d X9 = un EXPLIC
  si Z2 = 'PGTRT1' alors exec PGTRT1 sortie fin
  si Z2 = 'E-CREME' alors exec E-CREME sortie fin
  si Z2 = ...
  ...
  sinon
     i '/// PROGRAMME ABSENT'
  fin
fin
:fdef
```

Ce programme U-EXECUTION sera donc à modifier et à recompiler chaque fois qu'une nouvelle mise en correspondance devra être réalisée, correspondant à l'introduction d'un nouveau schéma de transaction (contrainte de la version V1-5 de Socrate).

4 - Exemples de programmes de traitement

Voir E-CREME, E-CRELIST, E-CREEX, § C-8.

C6 - LE MONITEUR DE GESTION DES TRANSACTIONS : ANAL

Rappelons qu'une transaction est définie comme une suite de mots transmis sur cartes perforées ou à partir d'un terminal, et non soumis à un formattage ; l'espace entre deux mots consécutifs peut être quelconque.

L'appel du programme ANAL permet de *saisir*, *valider* et *traiter* une suite de transactions, jusqu'à la rencontre du mot FIN ; la variable Z1 devra préalablement contenir un nom de Projet et Y21 aura la valeur 0 ou 2, selon que l'on travaillera en conversationnel ou en batch.

1 - Le paramètre projet : Z1

La nécessité de définir le projet sous lequel on veut voir traiter l'ensemble des transactions est explicitée au § C-7 concernant les primitives de gestion du dictionnaire.

Parmi les outils proposés dans NOYAU, un certain nombre a été lui-même défini par le mécanisme des schémas de transaction, ainsi ont été définis sous le projet NOYAU les schémas de transaction CREEX, CRELIST et CREME.

D'autres projets peuvent être créés par utilisation de la macro-instruction NPROJET :

par exemple :
L'exécution de NPROJET MACSI-P définit un nouveau projet MACSI-P sous lequel il sera possible de traiter des transactions.

2 - Le paramètre mode : Y21

Il est nécessaire de définir sous quel mode (batch ou conversationnel) on veut faire travailler le programme ANAL.

En cas d'erreur en effet, des dispositions différentes seront prises selon le mode choisi.

(1) En conversationnel (Y21 = 0), aucune erreur ne sera considérée comme fatale : la "conversation" en cas d'erreur sera reprise à partir du mot sur lequel l'erreur est détectée.

(2) En batch (Y21 = 2), certaines erreurs seront considérées comme fatales, tandis que pour d'autres la validation pourra se poursuivre sans toutefois que le traitement n'ait lieu.

Enfin, certains outils seront disponibles exclusivement en mode conversationnel : éditeur de texte, module pédagogique (cf. § C-6-6), etc.

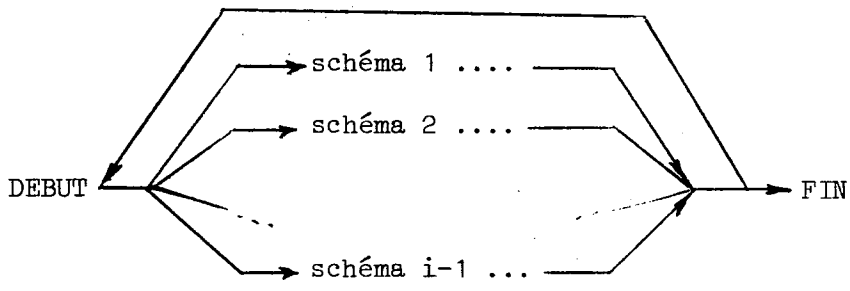
3 - Train de transactions

La saisie d'un train de transactions se fera donc par une séquence telle que :

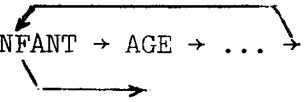
```
m Z1 = '...'
m Y21 = ...
exec ANAL  ?
[1ère transaction
[2ème transaction
.......
[dernière transaction
FIN
```

4 - Structure de stockage d'un schéma de transaction

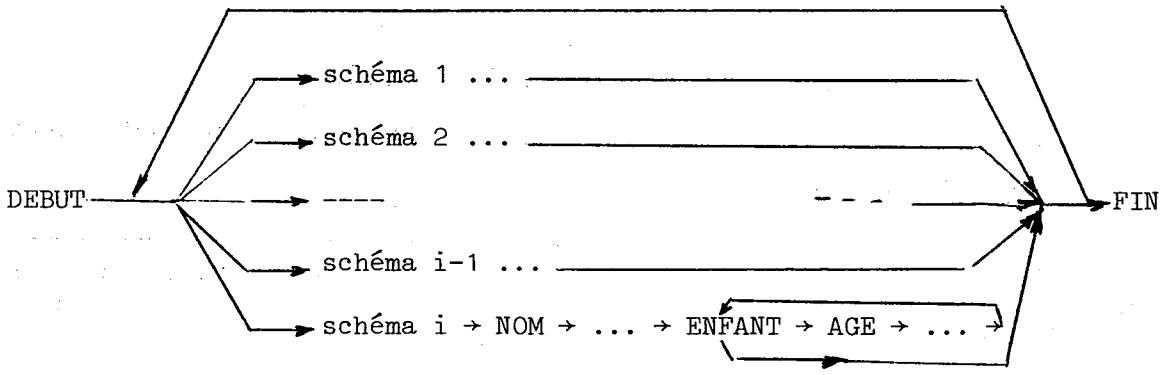
Un ensemble de schémas de transactions est encore un schéma de transaction. La description d'un schéma de transaction réalise en fait une ou plusieurs mises à jour de règles (cf. § C-3) qui permettent ainsi de compléter un réseau des schémas : l'introduction d'un nouveau schéma est alors l'addition d'un nouveau chemin à un réseau primitif. Soit le réseau de schémas :



La description du schéma i : NOM → ... → ENFANT → AGE → ... →



conduit au nouveau réseau :



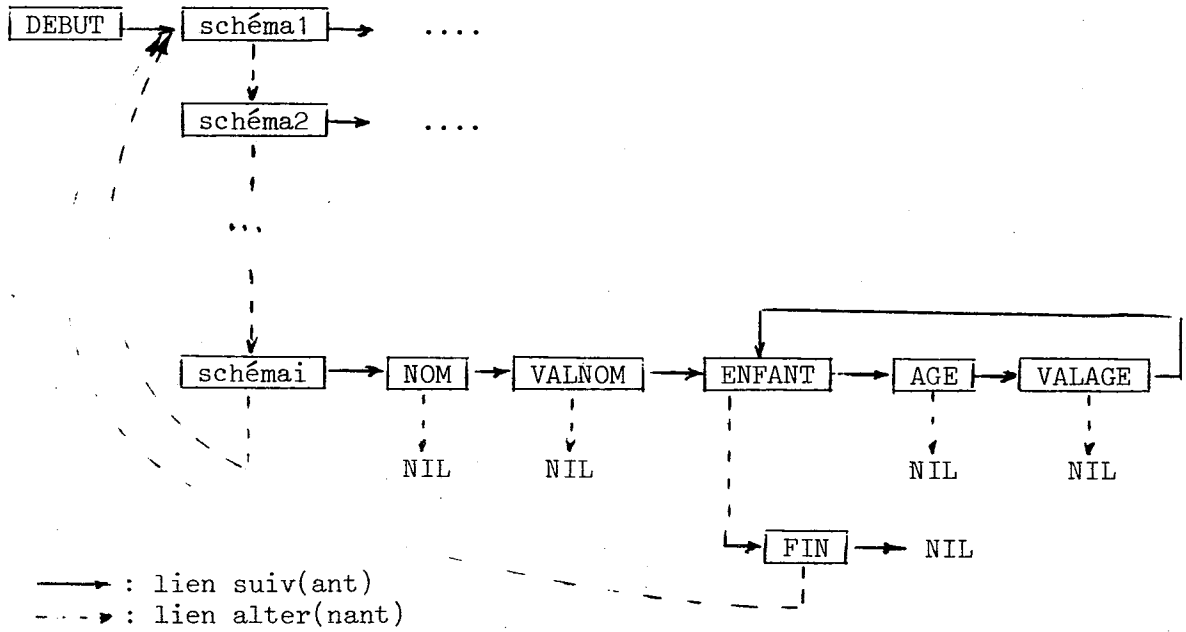
C'est en parcourant ce réseau que le moniteur ANAL gère la *saisie*, *le contrôle* et *l'exécution* d'une transaction réglée par un schéma particulier. Remarquons que chaque nouveau schéma a un nom (schéma 1, ...).

Ce réseau est stocké suivant une structure basée sur la technique "alternant-successeur" où chaque noeud du graphe correspond à la réalisation d'une entité ELEM (élément) :

entité ELEM
début
- NOM (nom symbolique du noeud)
- TYPE (clé, code, mot, num, ligne, texte ou liste)
- SUIV réfère un ELEM (successeur d'un élément)
- ALTER réfère un ELEM (élément alternant)
- SEMANT (nom d'un programme de contrôle)
- EXECUT (nom d'un programme de traitement)

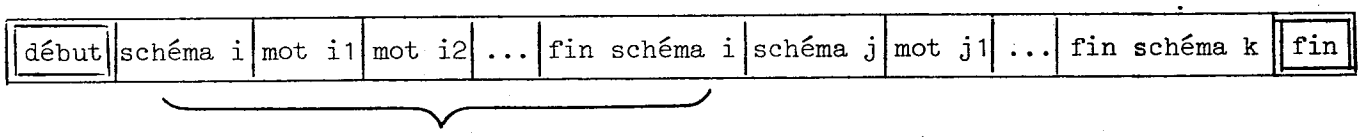
fin

Ainsi le réseau précédent serait enregistré :



5 - Algorithme général du moniteur central ANAL

Un train de transactions est considéré par le programme ANAL comme une suite de mots :



Une transaction commence par le nom du schéma suivant lequel elle est réglée.

```
programme ANAL :
   }
m X1 = un ELEM ayant NOM = 'début' ;
faire /* traitement d'un train de transactions */
  | si existe un suiv(ant) de X1 = 'fin' alors SORTIE fin
  | faire /* analyse (saisie + contrôle) d'une transaction */
  |    | lire mot suivant dans Z1
  |    | si EXISTE un suiv(ant) X2 ayant NOM = Z1;de X1
  |    |   | alors si TYPE de Z1 ≠ TYPE de X2 alors ERREUR ... fin
  |    |   |       M Z2 = SEMANT de X2
  |    |   |       si EXISTE Z2 alors CALL Z2
  |    |   |       |                    si ERREUR alors ... fin
  |    |   |       fin
  |    |   |       m X1 = X2
  |    |   |       si X2 = ELEM de fin de transaction alors SORTIE fin
  |    |   |       REFAIRE
  |    | fin
  |    | ERREUR ...
  |    | si X1 = ELEM de fin de transaction alors SORTIE fin
  |    | REFAIRE
  | fin
/* traitement d'une transaction */
  | si pas d'ERREURS alors M Z3 = EXECUT(ion) de X2
  |   |                        CALL Z3
  | fin
  | REFAIRE
fin
     }
```

6 - Le mode conversationnel pédagogique

En mode conversationnel, l'utilisateur effectue ses transactions à partir d'un terminal ; nous avons déjà spécifié qu'alors aucune erreur n'est considérée comme fatale, la conversation pouvant reprendre à partir de l'endroit où l'erreur s'est produite.

Les schémas de transactions étant documentés, l'utilisateur pourra alors, en tout moment d'une transaction, interroger le système pour savoir comment continuer sa transaction ; il dispose pour cela de fonctions d'aide :
une fonction d'aide est un mot réservé qui peut être inséré à n'importe quel endroit d'une transaction (excepté à l'intérieur d'un texte, ligne ou mot) ; ce mot réservé commence par un point.

1. La fonction .SOS

En un certain point d'une transaction, cette fonction permet à l'utilisateur d'apprendre du système ce qu'il est autorisé à faire (si plusieurs alternatives lui sont possibles) ou ce qu'il doit faire (au cas où il n'y aurait qu'une seule possibilité).

Effectuer une transaction consiste à se déplacer dans un graphe ; ce graphe est saisi et documenté : chaque noeud a reçu un code, un libellé et éventuellement un texte explicatif.

Appeler la fonction .SOS en un certain noeud aura alors pour effet d'imprimer sur le terminal le code et le libellé de tous les noeuds suivants ainsi que les contrôles qui, éventuellement, sont attachés à ces noeuds.

2. La fonction .AIDE (.FAID)

C'est une fonction .SOS permanente, c'est-à-dire un appel continu à la fonction .SOS, pour que le système précise à chaque instant comment l'utilisateur peut poursuivre la formulation de sa transaction ; cette aide permanente est annulée par la fonction .FAID.

3. La fonction .STOP

Elle permet à l'utilisateur d'abandonner la transaction en cours de transmission et de passer alors à une transaction suivante.

4. La fonction .EXPL

Alors que la fonction .SOS donne pour chaque noeud suivant, son code, libellé et éventuellement contrôle sémantique attaché , la fonction .EXPL suivie d'un code de noeud donne tous les textes explicatifs attachés à ce code

Cette même fonction, suivie cette fois du mot-clé ERREUR suivi lui-même d'un numéro d'erreur, donne le texte explicatif lié à cette erreur, tel qu'il a été défini au cours d'une transaction réglée par le schéma CREME (cf. § C-8-1).

5. La fonction .EDIT

Au cours d'une transaction, elle permet de connaître du système tous les mots saisis depuis le début de cette transaction.

7 - Liste des messages d'erreurs

Les messages d'erreurs édités par la macro-instruction ERREUR sont de la forme :

```
ERREUR <numéro> SUR MOT : <chaîne>  <libellé>
```

<numéro> est le code numérique de l'erreur
<chaîne> est la chaîne de caractères analysée
<libellé> est le texte explicatif de l'erreur.

Les erreurs propres au moniteur central sont :

1 : * ligne vide *

2 : * un code est limité à 12 caractères *

3 : * un code doit commencer par une lettre *

4 : * erreur * de début *
* un mot ou une ligne sont encadrés de 2 *

5 : * un mot est limité à 30 caractères *

6 : * une ligne est limitée à 60 caractères *

7 : * absence * de fin *
* cette valeur ne figure pas dans la liste prévue *

8 : * un texte est limité à 200 lignes *

10 : * valeur non numérique *

11 : * valeur inconnue dans une liste *
* une règle ne peut pas être mise à jour lorsque la précédente alternative ne commençait pas par un mot-clé, une ligne ou un texte *

27 : * mise à jour d'une règle inexistante *

28 : * la règle ne peut pas être mise à jour *
* le premier élément de la première mise à jour d'une règle ne peut être : NULL, GOTO, REG *

29 : * les éléments NULL, REG, GOTO sont prohibés *
* le programme de traitement d'une règle est défini lors de la mise à jour de la règle ANAL correspondante *

30 : * branchement à une règle inexistante *

31 : * programme de traitement d'une règle *
* ce code existe déjà associé à un objet de nature ou de projet différent *

32 : * initialisation (REG) d'une règle déjà existante *

35 : * liste de valeurs inexistante *

53 : * code interdit *
* le libellé d'un élément est précisé lors de la première désignation de cet élément *

54 : * erreur libellé *
* le message ne doit pas déjà exister *

57 : * code inexistant *

58 : * liste déjà définie *

59 : * valeur déjà définie *

61 : * message déjà défini *

C7 - LE DICTIONNAIRE CENTRALISE ET SES PRIMITIVES DE GESTION

Ce dictionnaire a pour but de rapprocher dans un même ensemble (les entités DICO et EXPLIC) tous les objets définis dans l'application que l'on se propose de gérer, que ces objets soient de même nature (appartiennent à une même entité SOCRATE), ou de nature différente.

Pour chaque élément de ce dictionnaire, on stockera donc le CODE de l'objet correspondant (le code d'un objet permet de nommer cet objet de façon unique), ainsi qu'éventuellement son LIBELLE et des TEXTES explicatifs.

Chaque élément du dictionnaire sera également défini par sa NATURE et son PROJET.

1 - "Nature" et "Projet" d'un objet

- Le paramètre "NATURE" permet de caractériser l'entité à laquelle appartient cet objet ; la gestion des natures d'objets est laissée à l'appréciation de l'analyste

(1 ≤ NATURE ≤ 200 avec NATURE = 8 si objet "programme"
= 31 si objet "mot clé"
= 40 si "liste de valeurs").

- Le paramètre "PROJET" :
NOYAU permet de gérer des transactions concernant plusieurs projets différents. Le moniteur d'enchaînement des transactions est en effet appelé avec le nom du projet sous lequel on veut travailler, pour paramètre ; à tous les objets définis au cours de cet appel, NOYAU associera ce projet ; aucune référence, d'autre part, ne pourra être faite à des objets définis sous d'autres projets.

2 - Les primitives de gestion du dictionnaire : U-AMAJ, U-INIT, U-LIB, U-BGEX, U-GLED

Ces primitives peuvent être appelées soit à partir des programmes de contrôle, pendant la phase de validation d'une transaction (U-INIT, U-AMAJ), soit à partir des programmes de traitement (U-LIB, U-BGEX, U-GLED). Ces primitives sont appelées sous un certain projet (cf. § C-7-1).

1. La primitive U-AMAJ

but	faire référence à un objet d'une certaine nature	
entrées	Z1	désigne un code d'objet
	Y11	désigne une nature d'objet
sorties	Y12 = {0, 1, 2}	. Y12=0 et X8 pointe le dico correspondant, s'il existe au dictionnaire un objet de ce projet ayant ce code Z1 et cette nature Y11 . Y12=1 et X8 pointe le dico correspondant, s'il existe un objet ayant ce code Z1 mais de nature différente ou pour un autre projet . Y12=2 et X8=U, s'il n'existe aucun objet ayant ce code Z1.
	X8	

2. La primitive U-INIT

but	faire référence à un objet existant ou le créer s'il est nouveau	
entrées	Z1	désigne un code d'objet
	Y11	désigne une nature d'objet
sorties	Y12 = {0, 1, 2}	. Y12 = 0 S'il existe dans le dictionnaire un élément ayant pour code Z1, s'il a la nature Y11 demandée et s'il est défini pour le projet actif, il sera alors pointé par X8 . Y12 = 1 S'il n'existe pas dans le dictionnaire d'objet ayant pour code Z1, alors un nouvel élément de DICO est créé, pointé par X8, caractérisé par cette nature Y11 et ce projet. Il est important de noter que cette création a lieu pendant la phase de validation d'une transaction ; si lors d'un contrôle ultérieur, cette transaction est rejetée, il sera alors nécessaire d'invalider toutes les créations effectuées, c'est-à-dire de restaurer le dictionnaire dans l'état où il se trouvait au début de la phase de validation ; cette nécessité est toutefois prise automatiquement en charge par NOYAU. . Y12 = 2 S'il existe dans le dictionnaire un élément ayant pour code Z1 (il sera alors pointé par X8) et s'il n'a pas la nature Y11 demandée ou s'il n'appartient pas au même projet.
	X8	

3. *La primitive U-LIB*

but	associer un libellé (chaîne alphanumérique de 60 caractères) à un élément du dictionnaire (défini par X8) au cours d'une phase de traitement	
entrées	X8	pointe le dico auquel on veut rattacher le libellé
	Z1	contient l'étoile de début de libellé
sorties	X8	pointe le dico auquel on a rattaché le libellé
	Z1	contient le mot qui suit l'étoile de fin de libellé

Exemple : ~~~ * ..... *une ligne de libellé* ..... * ~~~~

Z1 = [*] *avant appel*

Z1 = [~~~] *après appel*

4. *La primitive U-BGEX*

but	lire un texte (chaîne alphanumérique d'au plus 200 lignes de 60 caractères) et le ranger dans une entité EXPLIC (définie par X9) au cours d'une phase de traitement	
entrées	Z1	contient l'étoile de début du texte
sorties	X9	pointe l'explic qui vient d'être stockée
	Z1	contient le mot suivant l'étoile de fin de texte

cf. exemple § C-5-1.

5. *La primitive U-GLED*

but	associer un texte explicatif à un élément du dictionnaire	
entrées	X8	pointe un élément de DICO
	X9	pointe un texte de EXPLIC
sorties	/	l'explic X9 est chaînée au dico X8

Plusieurs explications peuvent ainsi être chaînées au même élément du dictionnaire.

3 - Autres caractéristiques du dictionnaire

D'autres caractéristiques sont définies dans l'entité DICO dont la gestion est laissée au choix de l'analyste :

- la caractéristique NUMENT de un DICO permettra, en particulier, de pointer l'objet auquel se réfère l'élément de DICO (ce pointage s'effectuant par numéro d'ordre) ; ainsi, si NATURE = 4 désigne l'entité PERSONNE, alors le "DICO ayant NATURE = 4 et NUMENT = 2" désignera la deuxième PERSONNE
- les caractéristiques SY1, SY2, SY3, SY4, SZ1, SZ2, SZ3 et SZ4 de un DICO n'ont aucune signification dans NOYAU : ce sont des variables qui pourront être utilisées librement par le concepteur du prototype.

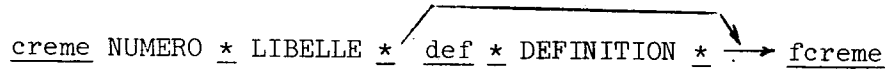
C8 - APPLICATION DE NOYAU A LUI-MEME

Un certain nombre d'outils disponibles dans NOYAU sont des schémas de transactions particuliers ; nous allons en décrire trois en donnant successivement leurs but, schéma, règles, programmes de contrôle, messages d'erreurs, programme de traitement et exemples, afin de permettre une bonne compréhension des paragraphes C-3, C-4, C-5 et C-7.

1 - Le schéma CREME

Pour décrire un message d'erreurs pour des programmes de contrôle (cf. § C4-7).

. schéma :



. description :

```
NOYEAU ?
MREG ANAL
CLE CREME *
CREATION DE MESSAGE *
NUM NUMEROM *
NUMERO DE MESSAGE *
SEMANT S-CREME *
LE MESSAGE NE DOIT PAS DEJA EXISTER *
LIGNE LIBMESS *
LIBELLE DU MESSAGE *
REG DEFM
CLE FCREME *
FIN DE CREATION DE MESSAGE *
FREG EXEC E-CREME

MREG DEFM
CLE DEF
TEXTE DEFMESS *
EXPLICATION DU MESSAGE *
FFREG
FIN
```

. le programme de contrôle S-CREME :

```
:DEFPRO S-CREME :CONTXT XO D X8 = UN DICO :EXP
M Y1 = Z1
SI EXISTE UN MESSAGE Y1 ALORS ERREUR (61) FIN
:FDEF ?
```

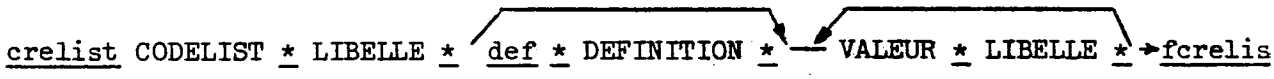
. le programme de traitement E-CREME :

```
:DEFPRO E-CREME :CONTXT X0
      D X8 = UN DICO D X9 = UN EXPLIC :EXP
LEC 1
M Y1 = Z1
G UN MESSAGE Y1
M X1 = UN MESSAGE Y1
LEC 2
M DEB-LIBEL DE X1 = Z1
LEC 1
M FIN-LIBEL DE X1 = Z1
LEC 2
SI Z1 = 'DEF' ALORS
   LEC 1
   EXEC U-BGEX
   M DEFINITION DE X1 = X9
FIN
D X1
:FDEF ?
```

2 - Le schéma CRELIST

Pour décrire une liste de valeurs.

. schéma :



. description :

```
NOYEAU ?
MREG ANAL
CLE CRELIST *
CREATION DE LISTE *
CODE CODELIST SEMANT S-CRELIST *
CETTE LISTE NE DOIT PAS DEJA EXISTER *
LIGNE LIBLIST *
LIBELLE DE LA LISTE *
REG DEFLIST
REG DEFVAL
FREG EXEC E-CRELIST

MREG DEFLIST
CLE DEF
TEXTE DEFLIST *
EXPLICATION DE LA LISTE *
FFREG

MREG DEFVAL
CLE FCRELIST *
FIN DE CREATION DE LISTE *
FREG

MREG DEFVAL
NUM VALELEM *
VALEUR DE L'ELEMENT *
SEMANT S-VALELEM *
CETTE VALEUR DOIT ETRE COMPRISE ENTRE 1 ET 100 *
LIGNE LIBVAL *
LIBELLE D'UNE VALEUR DANS UNE LISTE *
GOTO DEFVAL
FREG
FIN
```

. les programmes de contrôle S-CRELIST et S-VALELEM et les messages d'erreurs :

```
:DEFPRO S-CRELIST :CONTXT XO D XR = UN DICO :EXP
M Y11 = 40 EXEC U-INIT
SI Y12 ≠ 1 ALORS ERREUR (58) FIN
:FDEF ?

:DEFPRO S-VALELEM :CONTXT XO D XR = UN DICO :EXP
SI Y1 < 1 OU Y1 > 100 ALORS ERREUR (59) FIN
:FDEF ?
```

```
CREME 58 *
LISTE DEJA DEFINIE *
FCREME
CREME 59 *
VALEUR DEJA DEFINIE *
FCREME
FIN
```

. le programme de traitement E-CRELIST :

```
:DEFPRO E-CRELIST ;CONTXT X0
D X8 = UN DICO D X9 = UN EXPLIC ;EXP
D X4 = UN CONTENU DE UN EXPLIC
G UN LISTVAL X1 M Y1 = NUMDE X1
LEC 1
M NUMENT DE UN DICO X8 AVEC CODE = Z1 ; = Y1
POUR X1
   LEC 2
   M DEB-LIBEL = Z1
   LEC 1
   M FIN-LIBEL = Z1
   LEC 2
   SI Z1 = 'DEF' ALORS
      LEC 1
      POUR X0 EXEC U-BGEX EXEC U-GLED FIN
   FIN
   M Y1 = NUM-EX
   M Y1 = Y1 + 1
   POUR X0 M NUM-EX = Y1 FIN
   G UN EXPLIC X9 DE X0
   M DEFINITION = X9
   POUR X9
      M CODE = Y1 M SIGNAL = 1
      M PROJET = Y24
      G UN CONTENU X4
      POUR X4
         M D-L = 'CETTE CARACTERISTIQUE PEUT PRE'
         M F-L = 'NDRE LES VALEURS SUIVANTES , '
      FIN
      G UN CONTENU X4
      POUR X4
         M D-L = 'AVEC LA SIGNIFICATION CORRESPO'
         M F-L = 'NDANTE :'
      FIN
   FIN
   FAIRE
      G UN CONTENU X4 DE X9
      M D-L DE X4 = Z1
      M Y1 = Z1
      G UN ELEMENT Y1 M X2 = UN ELEMENT Y1 DE X1
      G UN CONTENU X4 DE X9
      POUR X2
         LEC 2
         M DEB-LIBEL = Z1
         M D-L DE X4 = Z1
         LEC 1
         M FIN-LIBEL = Z1
         M F-L DE X4 = Z1
      FIN
```

```
        LEC 2
        SI Z1 = 'FCRELIST' ALORS REFAIRE FIN
    FIN
FIN
D X1 D X2 D X4
:FDEF ?
```

3 - Le schéma CREEX

Pour associer un même texte explicatif à plusieurs éléments du dictionnaire.

. schéma :

creex ← CODELEM — * DEFINITION * → fcreex

. description :

```
NOYEAU ?
MREG ANAL
CLE CREEX *
CREATION D'EXPLICATION LIEE A PLUSIEURS ELEMENTS *
CODE CODELEM *
CODE D'ELEMENT *
SEMANT S-CREEX *
LE CODE DOIT ETRE CELUI D'UN ELEMENT EXISTANT DANS LA BASE *
REG BOUCLE
FREG EXEC E-CREEX

MREG BOUCLE
TEXTE EXPLIC *
EXPLICATION LIEE A PLUSIEURS ELEMENTS DE LA BASE *
CLE FCREEX *
FIN DE CREATION D'EXPLICATION *
FREG

MREG BOUCLE
CODE CODELEM SEMANT S-CREEX
GOTO BOUCLE
FREG
FIN
```

. le programme de contrôle S-CREEX et messages d'erreurs :

```
:DEFPRO S-CREEX :CONTXT X0 D X8 = UN DICO :EXP
M X8 = UN DICO AVEC CODE = Z1 ;
SI PAS X8 ALORS
   ERREUR (57)
SINON
   SI PROJET DE X8 = Y24 ALORS ERREUR (53) FIN
FIN
:FDEF ?

NOYEAU ?
CREME 57 *
CODE INEXISTANT *
FCREME
FIN
```

. le programme de traitement E-CREEX :

```
:DEFPRO E-CREEX :CONTXT X0
        D X8 = UN DICO D X9 = UN EXPLIC :EXP
FAIRE
     LEC 1
     SI Z1 = '*' ALORS REFAIRE FIN
     EXEC U-BGEX
FIN
M Y18 = 1
FAIRE
     LEC 1
     SI Z1 = '*' ALORS
        M X8 = UN DICO AVEC CODE = Z1 ;
        EXEC U-GLED
        REFAIRE
     FIN
FIN
:FDEF ?
```

C9 - PROGRAMMATION DES DOSSIERS DE SORTIE

1 - Présentation d'un dossier de sortie

Il consiste pour un projet donné à sortir tout ou une partie de la documentation définie pour ce projet.

Un dossier de sortie est un listing ; il comprendra plusieurs chapitres : par exemple, un premier chapitre définissant le plan du dossier, suivi d'un chapitre pour chaque nature d'objets à éditer, terminé éventuellement par un chapitre donnant un glossaire de tous les objets listés avec des renvois aux pages où l'objet correspondant est édité.

En tête de chaque chapitre, pourra figurer une page de garde où figure le nom du chapitre ; en haut et en bas de chaque page, figureront le numéro de cette page ainsi que le nom du chapitre en cours d'édition, ainsi peut-être que le nom de l'objet qui y est décrit.

Ainsi, peut-on parler de dossier "standard" de sortie, pour tout dossier basé sur de tels principes. L'écriture du programme correspondant est alors basé sur un certain nombre d'outils que nous décrivons dans ce chapitre.

Un résumé synthétique de la hiérarchie des appels des différents outils est donné au § C12-3.

2 - Sortie d'un dossier pour un projet donné : DOSANA, SEDT, DOSSIER

On exécutera la macro-instruction DOSANA avec le nom du projet pour paramètre , le nom du programme qui sélecte les objets à lister et le nom du programme qui définit les différents chapitres du dossier (en particulier, pour chaque nature possible d'objets à lister, le programme d'édition correspondant).

1. Définition du programme DOSANA

```
:defmac DOSANA : : : :exp
exec DATSYS                          /* Z7 : date du jour */
m Z4 = :1:                           /* Z4 : nom du projet */
faire
| m X4 = un PROJET ayant NOM = Z4 ;
| si pas X4 alors ERREUR (114) sortie fin
| m Y23 = numde X4
| d X4                               /* Y23 : numéro du projet */
| m Y24 = 0                          /* Y24 = compteur de pages */
| m Y21 = 0                          /* Y21 : compteurs de lignes */
| m Z5 : 'DOSSIER D'ANALYSE'         /* Z5 : nom du chapitre */
| CADRE
| exec :2:                          /* :2: = programme de sélection */
| exec :3:                          /* :3: = programme d'édition */
fin
:fdef
```

2. Le programme de sélection des objets à lister : SEDT

- C'est le deuxième paramètre d'appel de la macro DOSANA.
- Pour chaque NATURE d'objets à lister, ce programme devra définir un inverse sur l'entité DICO des réalisations qu'il veut voir lister pour cette nature ; il disposera pour cela de l'entité

```
entité 200 INVEDIT
|ELEMENT inverse tout DICO
```

  (NOYAU permet d'envisager un maximum de 200 natures différentes d'objets ; chaque réalisation de INVEDIT sera donc accédée par son NUMDE).
- Ce programme ne devra pas modifier les variables (y21, y23, y24, Z4, Z7) initialisées par DOSANA ; les éléments du dictionnaire accessibles pour ce projet sont nommés par :

```
DICO ayant PROJET = y23
```

- Un programme standard de sélection, SEDT, est mis à la disposition de l'analyste qui demande l'édition de tous les objets liés au projet :

```
:defpro SEDT :contxt X0 :exp
   |pour tout DICO X8 ayant PROJET = y23 ;
   |   m y15 = NATURE de X8
   |   g un ELEMENT de un INVEDIT y15 = X8
   |fin
:fdef
```

3. *Le programme du dossier proprement dit : DOSSIER*

- C'est le troisième paramètre du programme DOSANA, il définit les différentes phases du dossier ; par exemple :

```
:defpro DOSSIER :contxt X0 :exp
   |        /* introduction */
   |  m Z5 = 'PLAN DU DOSSIER'
   |  CADRE
   |  exec PLAN
   |        /* premier chapitre */
   |  XEDS (MODULE) 4 EDIMO
   |        /* deuxième chapitre */
   |  XEDS (ACTION) 7 EDIAC
   |        /* troisième chapitre */
   |  XEDS (COMMANDE) 6 EDICO
   |  ....
   |        /* derniers chapitres */
   |  exec GLOSSAIRE
:fdef
```

4. *La macro XEDS*

Elle correspond à un chapitre ; elle a pour paramètres d'entrée :

1) le nom à donner à ce chapitre
2) la NATURE des objets à lister dans ce chapitre
3) le nom du programme d'édition à appliquer à chacun des objets sélectionnés de cette nature.

Elle produit une page d'en-tête pour ce chapitre et édite, à partir du programme mentionné, tous les objets sélectionnés de cette nature.

Elle est ainsi définie :

```
:defmac XEDS ( : ) : : :exp
| m Z5 = ':1:'
| CADRE
| faire
| | si pas un ELEMENT de un INVEDIT :2: alors
| | | HP
| | | i 'PAS DE REALISATION'
| | | BP
| | | sortie
| | fin
| | d X9 = une EXPLIC
| | pour tout ELEMENT par CODA ;
| | | de un INVEDIT :2:
| | | m Z6 = CODE de X3
| | | m ND1 de X8 = Y24
| | | HP
| | | pour X0 exec :3: fin
| | | BP
| | fin
| fin
:fdef
```

- La caractéristique ND1 de un DICO :
  . stocke le numéro de page où est décrit l'élément correspondant du dictionnaire
  . est accédée par l'utilitaire GLOSSAIRE qui peut être appelé en fin de dossier.

- La caractéristique CODA de un DICO.
  Elle permet d'accéder séquentiellement au dictionnaire, par ordre alphabétique croissant.

5. *La macro CADRE*

Elle permet de construire une page d'en-tête pour un chapitre ; Z5 contient le nom à donner à ce chapitre ; ce nom pourra être repris en haut et en bas de chaque page, grâce aux utilitaires HP et BP.

3 - Programme d'édition pour une nature donnée d'objets

- C'est le troisième paramètre de la macro XEDS.
- Il définit l'édition d'un objet d'une nature donnée.
- Il pourra utiliser les outils suivants (outils de mise en page : ECRIT, BL, HP, BP et outils d'édition de la documentation : EDLICO, EDTXT).

1. La macro ECRIT

Elle est utilisée à la place de l'instruction SOCRATE : ECRIRE ; elle compte d'autre part le nombre de lignes écrites à l'intérieur d'une page et effectue automatiquement les sauts de page avec impression d'un en-tête en haut et en bas de chaque page.

Dans cet en-tête, on verra apparaître l'information :

PAGE y24 Z7 Z4 Z5 : Z6

où

Y24 est le numéro de la page
Z7, la date du jour
Z4, le nom du projet
Z5, le nom du chapitre
Z6, le nom de l'objet en cours d'édition.

2. La macro BL

Elle remplace la macro ECRIT lorsque la (ou les) ligne à imprimer est blanche ; elle est utilisée avec un paramètre : BL n, qui indique le nombre de lignes blanches à imprimer.

3. La macro BP

Elle effectue un positionnement en bas de page, avec saut du nombre de lignes nécessaires et impression de l'en-tête.

4. La macro HP

Elle effectue le positionnement en haut de la page suivante, avec impression de l'en-tête.

5. *La macro EDLICO*

- Sur une ligne donnée, elle permet de commander l'impression à partir d'une certaine colonne, du code, puis du libellé d'un certain élément du dictionnaire.
- Elle est appelée avec deux paramètres : le premier précise la position dans la ligne, le second est un Xi : il précise le DICO sur lequel on applique cette commande.

6. *La macro EDTXT*

- Sur une ligne donnée, elle commande l'impression de tous les textes explicatifs associés à un élément donné de dictionnaire. Pour chaque texte édité, elle imprime son numéro d'ordre.
- Elle est appelée avec deux paramètres : le premier précise la position dans la ligne, le second est un Xi, il précise le DICO sur lequel on applique cette commande.
- Elle modifie les valeurs des variables Y14, Y15 et Y25.

7. *Contraintes sur les variables Xi, Yi, Zi*

- Nous avons vu la signification particulière des variables Z7, Z4, Z5, Z6, 21, Y23 et Y24.
- De ces variables, seules Z5 et Z6 peuvent être modifiées ; leur signification a été indiquée en ce qui concerne les en-têtes de page.

4 - Le glossaire du dossier : GLOSSAIRE

Un programme est à la disposition de l'utilisateur ; il lui permet en fin de dossier de commander l'édition d'un glossaire de ce dossier ; ce glossaire comprend deux parties :

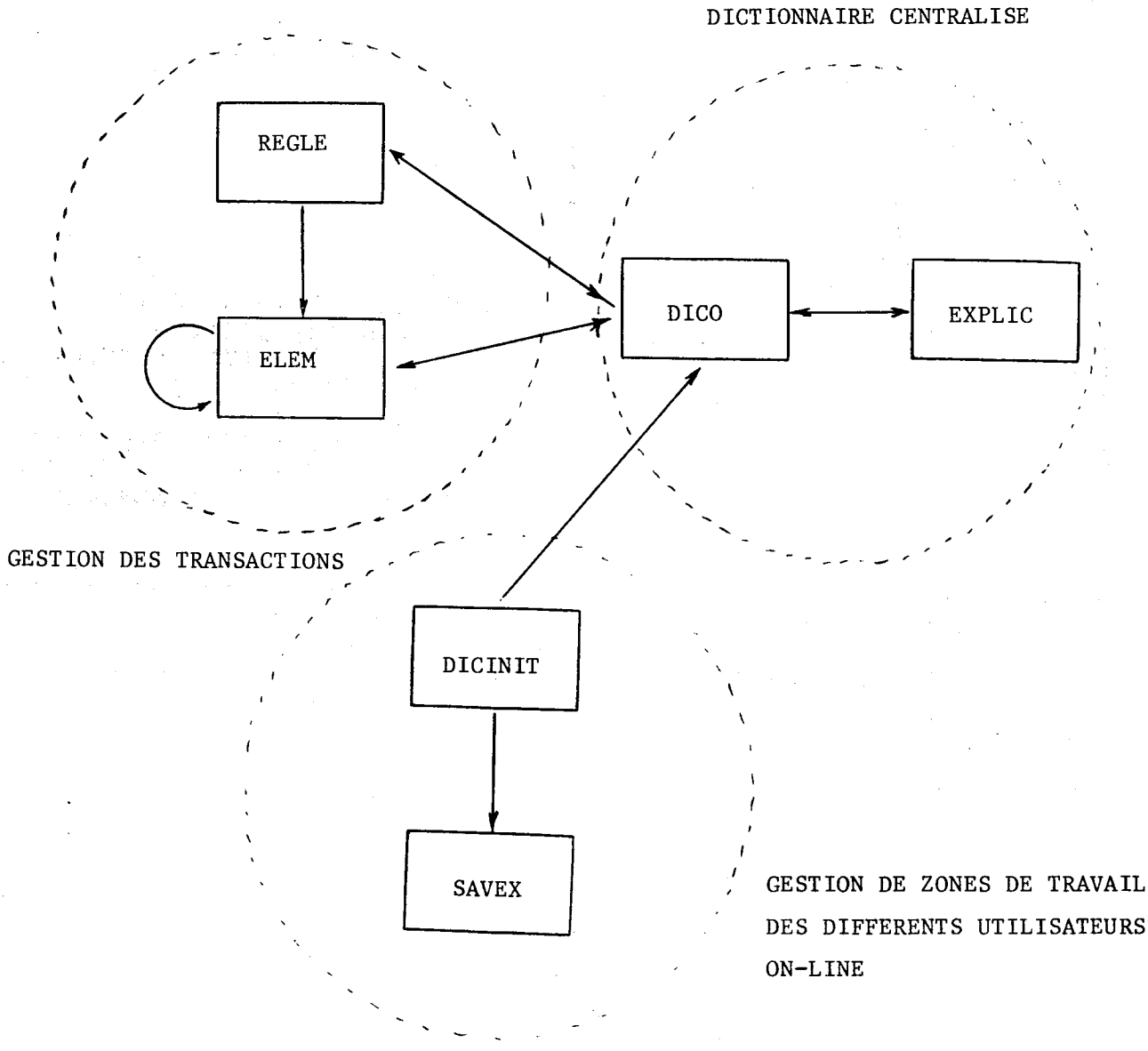
1) liste par ordre alphabétique des codes édités avec leur libellé et le numéro de page où ils sont décrits.
2) liste alphabétique des codes avec leur libellé et toutes les explications associées.

Ce programme GLOSSAIRE est ainsi conçu :

```
:defpro GLOSSAIRE :contxt X0 :exp
   | SE tout CODE-A-LISTER
   | pour tout ELEMENT X8 de tout INVEDIT
   |    g un CODE-A-LISTER = X8
   | fin
   |      PEDS        /* première partie */
   |      REDS        /* deuxième partie */
:fdef
```

C10 - STRUCTURE DOCUMENTAIRE DE NOYAU

1 - Modèle général



2 - Structure Socrate

```
DEBUT
      /* STRUCTURE DU NOYAU                                          */

DATE MOT 30
NUM-EX DE 0 A 15000                          /* NOMBRE COURANT D'EXPLIC */
CODE-A-LISTER INVERSE TOUT DICO              /*

/*-------------------------------------------------------------------*/
      /* L'ENTITE ELEM DECRIT LE LANGAGE D'APPLICATION               */
      /* UN ELEM EST UN ELEMENT TERMINAL DE LA GRAMMAIRE             */
      /* L'ENCHAINEMENT DES TERMINAUX DECRIT LA GRAMMAIR             */

ENTITE 4000 ELEM
DEBUT
   SUIV REFERE UN ELEM                       /* SUIVANT             */
   ALTER REFERE UN ELEM                      /* ALTERNANT           */
   NOM MOT 12                                /* IDENTIFIANT         */
   TYPE DE 30 A 100
          /* MOT-CLE          : 31    */
          /* CODE             : 35    */
          /* MOT              : 36    */
          /* LIGNE            : 37    */
          /* TEXTE            : 38    */
          /* NUMERIQUE        : 39    */
          /* LISTE DE VALEURS : 40  VOIRE LISTE-OU-NOMENC */
   LISTE-OU-NOMENC DE 1 A 100                /* NUMDE LISTVAL       */
   SEMANT DE 1 A 3000                        /* NUMDE DICO ,SPECIFIE UN */
                                             /* PROGRAMME DE CONTROLE */
                                             /* SUR L'ELEM          */
   EXECUT DE 1 A 3000                        /* NUMDE DICO ,SPECIFIE LE */
FIN                                          /* PROGRAMME D'EXECUTION */
                                             /* DE LA REGLE ASSOCIEE */

/*-------------------------------------------------------------------*/
      /* L'ENTITE REGLE DECRIT LES ELEMENTS NON                      */
      /* TERMINAUX DE LA GRAMMAIRE . UNE REGLE EST                   */
      /* UN ENSEMBLE DYNAMIQUE DE PRODUCTIONS                        */

ENTITE 1000 REGLE
DEBUT
   PREMIER REFERE UN ELEM                    /* PREMIERE PRODUCTION */
   DERNIER REFERE UN ELEM                    /* DERNIERE PRODUCTION; SI */
                                             /* INDEFINI,LA REGLE EST */
                                             /* FIGEE               */
   SORT REFERE UN ELEM                       /* SUIVANT DES PRODUCTIONS */
```

```
   NOM MOT 12 AVEC CLE UNIQUE FIN                    /* IDENTIFIANT          */
FIN

/*-------------------------------------------------------------------*/
      /* L'ENTITE LISTVAL DECRIT LES LISTES DE       */
      /* VALEURS NUMERIQUES                          */

ENTITE 200 LISTVAL                                   /* ACCES PAR NUM DE */
DEBUT
   DEB-LIBEL MOT 30 FIN-LIBEL MOT 30                 /* LIBELLE              */
   DEFINITION REFERE UNE EXPLIC                      /* TEXTE-SYSTEME        */
   ENTITE 200 ELEMENT                    /* ACCES PAR NUMDE OU PAR NOM */
   DEBUT                                             /* POUR CHAQUE VALEUR   */
      DEB-LIBEL MOT 30 FIN-LIBEL MOT 30              /* LIBELLE              */
      NOM MOT 30                                     /* POUR RETOUR-SEMANTIQUE */
      CODE DE 1 A 100                                /* VARIABLE DE TRAVAIL  */
   FIN
FIN

/*-------------------------------------------------------------------*/
      /* L'ENTITE DICO REGROUPE LES OBJETS DE LA BASE */

ENTITE 3000 DICO
DEBUT
   CODE MOT 12 AVEC CLE UNIQUE FIN                   /* IDENTIFIANT ( = CODA) */
   CODA MOT 12 AVEC CLE ORDONNE (4096 $SHCLAS) FIN
   DEB-LIBEL MOT 30 FIN-LIBEL MOT 30                 /* LIBELLE              */
   NATURE DE 1 A 200
   NUMENT DE 1 A 20000                               /* NUMDE L'ENTITE       */
   PROJET DE 1 A 7
   SY1 DE 0 A 65530                                  /* VARIABLE DE TRAVAIL  */
   SY2 DE 0 A 65530                                  /* VARIABLE DE TRAVAIL  */
   SY3 DE 0 A 65530                                  /* VARIABLE DE TRAVAIL  */
   SY4 DE 0 A 65530                                  /* VARIABLE DE TRAVAIL  */
   SZ1 MOT 30                                        /* VARIABLE DE TRAVAIL  */
   SZ2 MOT 30                                        /* VARIABLE DE TRAVAIL  */
   SZ3 MOT 30                                        /* VARIABLE DE TRAVAIL  */
   SZ4 MOT 30                                        /* VARIABLE DE TRAVAIL  */
   LISTEXTE REFERE UNE EXPLIC                        /* DEFINITION DE L'OBJET */
   ND1 DE 0 A 16000 ND2 DE 0 A 16000                 /* VARIABLES EDITION    */
FIN

/*-------------------------------------------------------------------*/
      /* DICINIT : ZONE DE TRAVAIL UTILISATEUR       */

ENTITE 10 DICINIT                                    /* ACCES PAR NUMDE : Y22 */
DEBUT
```

```
   SZ0 MOT 30                                    /* VARIABLE SYSTEME          */
   SZ1 MOT 30                                    /* VARIABLE DE TRAVAIL       */
   SZ2 MOT 30                                    /* VARIABLE DE TRAVAIL       */
   SZ3 MOT 30                                    /* VARIABLE DE TRAVAIL       */
   SZ4 MOT 30                                    /* VARIABLE DE TRAVAIL       */
   SZ5 MOT 30                                    /* VARIABLE DE TRAVAIL       */
   SZ6 MOT 30                                    /* VARIABLE DE TRAVAIL       */
   SY0 DE 0 A 65530                              /* VARIABLE SYSTEME          */
   SY1 DE 0 A 65530                              /* VARIABLE DE TRAVAIL       */
   SY2 DE 0 A 65530                              /* VARIABLE DE TRAVAIL       */
   SY3 DE 0 A 65530                              /* VARIABLE DE TRAVAIL       */
   SY4 DE 0 A 65530                              /* VARIABLE DE TRAVAIL       */
   SY5 DE 0 A 65530                              /* VARIABLE DE TRAVAIL       */
   SY6 DE 0 A 65530                              /* VARIABLE DE TRAVAIL       */
   TDI1 INVERSE TOUT DICO                        /* INVERSE DE TRAVAIL        */
   TDI2 INVERSE TOUT DICO                        /* INVERSE DE TRAVAIL        */
   TDI3 INVERSE TOUT DICO                        /* INVERSE DE TRAVAIL        */
   ELEM-EN-COURS DE 1 A 4000  /* POUR MODE PEDAGOGIQUE */
   DICO-NOU INVERSE TOUT DICO                    /* VARIABLE SYSTEME          */
   ENTITE 5000 STOCK
      /* STOCKAGE TEMPORAIRE D'UNE REQUETE UTILISATEUR */
   DEBUT                                         /* ACCES PAR NUMDE : Y18     */
      VAL MOT 30                                 /* MOT LU STOCKE             */
   FIN
FIN


/*-------------------------------------------------------------------------*/
      /* UN SAVEX EST UNE ZONE TEMPORAIRE POUR LES */
      /* EDITIONS DE TEXTE D'UN UTILISATEUR        */

ENTITE 10 SAVEX                                  /* ACCES PAR NUMDE : Y22     */
DEBUT
   REFEX REFERE UNE EXPLIC                       /* TEXTE EN MODIFICATION     */
   LINCOUR REFERE UNE LIGNE DE UN SAVEX          /* LIGNE COURANTE            */
   CODEX DE 1 A 15000                            /* NUMDE L'EXPLIC-REFEX      */
   ENTITE 202 LIGNE
   DEBUT
      PTR-AV REFERE UNE LIGNE DE UN SAVEX        /* LIGNE SUIVANTE            */
      PTR-AR REFERE UNE LIGNE DE UN SAVEX        /* LIGNE PRECEDENTE          */
      D-L MOT 30 F-L MOT 30                      /* VALEUR DE LA LIGNE        */
   FIN
FIN


/*-------------------------------------------------------------------------*/
      /* UN DICO2 CHAINE UN DICO SUR LES EXPLICS ASSOCIEES */

ENTITE 3000 DICO2                                /* ACCES PAR NUMDE : NUMDE   */
DEBUT                                            /*       DICO ASSOCIE        */
```

```
   ENTITE 100 EXPL-CHAINE
   DEBUT
      EXPL REFERE UNE EXPLIC
   FIN
FIN


/*-------------------------------------------------------------------*/
      /* L'ENTITE MESSAGE DEFINIT LES ERREURS                        */

ENTITE 2000 MESSAGE                          /* ACCES PAR NUMDE :NUMERO */
DEBUT                                        /*       DE L'ERREUR       */
   DEB-LIBEL MOT 30 FIN-LIBEL MOT 30         /* LIBELLE                 */
   DEFINITION REFERE UNE EXPLIC              /* TEXTE EXPLICATIF        */
   PROJET DE 1 A 7                           /* GRAMMAIRE ASSOCIEE      */
FIN


/*-------------------------------------------------------------------*/
      /* LES EXPLIC STOCKENT DES TEXTES                              */
      /* EXPLIC 1  EST LE TEXTE : RAS                                */

ENTITE 10000 EXPLIC
DEBUT
   CODE DE 1 A 15000 AVEC CLE UNIQUE FIN
   SIGNAL DE 1 A 2                           /* CLE PROTECTION EN MULTI */
   PROJET DE 1 A 7
   ENTITE 200 CONTENU                        /* LIGNES DU TEXTE , PAR   */
   DEBUT                                     /*     ORDRE CROISSANT     */
      D-L MOT 30 F-L MOT 30                  /* VALEUR DE LA LIGNE      */
   FIN
   ENTITE 100 DICO-CHAINE                    /* DICOS CONCERNES PAR     */
   DEBUT                                     /*      CETTE EXPLIC       */
      MOT-CLE REFERE UN DICO
   FIN
FIN


/*-------------------------------------------------------------------*/
      /*  ENTITE DE SELECTION POUR LES EDITIONS.                     */
      /*                                                             */

ENTITE 200 INVEDIT
DEBUT
   ELEMENT INVERSE TOUT DICO
FIN


/*-------------------------------------------------------------------*/
      /* CARTE : POUR TRAVAIL SUR ENREG DE 80 CARAC */
```

```
FORMAL CARTE
DEBUT
   FORMAL 80 CARAC
   DEBUT
      CAR MOT 1
   FIN
   KARTE MOT (1 80)
   KCODE MOT (1 12)
   KDEB  MOT (1 30)
   KFIN  MOT (31 30)
   KAR1  MOT (1 1)
   KAR31 MOT (31 1)
   KUTIL MOT (1 60)
   KTROP MOT (61 20)
FIN

/*-------------------------------------------------------------*/
      /* LIGNE : POUR TRAVAIL SUR ENREG DE 133 CARAC */

FORMAL LIGNE
DEBUT
   FORMAL 133 CARAC
   DEBUT
      CAR MOT 1
   FIN
   SAUC MOT (1 1)
   LIG1 MOT (2 30)
   LIG2 MOT (32 30)
   LIG3 MOT (62 30)
   LIG4 MOT (92 30)
   LIG5 MOT (122 11)
FIN

/*    FIN DES STRUCTURES DE TRAVAIL UTILISEES PAR MACSI-P.      */

FIN
```

C11 - LISTE DES PROGRAMMES (MACROS-INSTRUCTIONS ET PROGRAMMES PRECOMPILES)

```
** SOCRATE STARTED **  V1.5     LE: 04/02/77  A 11.34.35.86.
                  1    :LISALL ?
I SHCLAS      02/06/75153          1
T SAUT        02/06/75153          1
I LECFLOT     04/06/75155          1
T LECLIN      04/06/75155          1
C U-EDICO                          2
C U-IMPR                           1
          C U-EDICO
C U-ERR                            1
M ERREUR                           1
          C U-ERR
M DY22                             1
C U-EXPLS                          2
          M DY22
          C U-EDICO
C K-SPECIAUX                       4
          C U-EXPLS
          I LECFLOT
          C U-EDICO
          M DY22
          C U-IMPR
C K-LEC                            1
          C U-EXPLS
          I LECFLOT
          C K-SPECIAUX
C U-INIT                           2
          M DY22
C U-AMAJ                           1
M ABANDONLIGNE                     1
C K-LIN                            1
          I LECLIN
M SUIVAN                           1
M ERRNONFATAL                      1
          M DY22
          M SUIVAN
          M ABANDONLIGNE
C I-LEC                            1
          I LECFLOT
C I-MREG                           7
          C I-LEC
C U-RANGE                          1
          M DY22
M LEC                              1
C U-GLED                           3
C U-BGEX                           4
          M LEC
C S-CODREG                         1
          M ERREUR
C S-NUMREGOTO                      1
          M ERREUR
C S-GOTOREG                        1
          M ERREUR
C S-FREG                           1
M U-SLYBELEM                       1
          C U-INIT
          M ERREUR
C S-REGCOD                         1
```

```
C S-CODE                                1
          M U-SLIBELEM
C S-CODOP                               1
          M U-SLIBELEM
C S-CODELIST                            1
          C U-INIT
          M ERREUR
C E-MREG                                11
          M LEC
          C U-INIT
          C U-RGEX
          C U-GLED
C S-CREME                               1
          M ERREUR
C E-CREME                               3
          M LEC
          C U-RGEX
C S-CRELIST                             1
          C U-INIT
          M ERREUR
C S-VALELEM                             1
          M ERREUR
C E-CRELIST                             6
          M LEC
          C U-RGEX
          C U-GLED
C E-CREEX                               2
          M LEC
          C U-RGEX
          C U-GLED
I CHANGE      23/06/75174               1
I CHERCHE     23/06/75174               1
I LECHAIN     23/06/75174               1
I LECPARM     23/06/75174               1
C ED-REST                               4
C ED-I                                  2
          I LECLIN
C ED-BT                                 1
C ED-SAUVE                              3
C ED-R                                  3
          I LECPARM
          I CHANGE
C ED-E                                  4
          I LECHAIN
          I CHERCHE
C ED-L                                  3
          I LECHAIN
          I CHERCHE
C ED-S                                  2
          I LECHAIN
          I CHERCHE
C ED-P                                  2
          I LECHAIN
          I CHERCHE
C EDEX                                  4
          C ED-REST
C E-EDEX                                1
C U-EXECUTION                           1
          C F-MREG
          C F-CREME
          C F-CREEX
          C F-CRELIST
```

```
                C F-FDEX
I LECLIN1     10/02/76041          1
C K-LIN1                           1
          I LECLIN1
C SEDT                             1
C GLOSSAIRE                        44
             M PEDS
             M REDS
I DATSYS      30/09/75273          1
M EDSTANDART                       18
             M ECRIT
             M RL
             M FDLIB
             M FDTXT
M LTIRE                            2
             M ECRIT
M EDINVERSE                        5
             M EDLICO
             M RL
C S-CREEX                          1
             M ERREUR
C U-SEMAN                          2
             C S-CREEX
             C S-NULREGOTO
             C S-CODREG
             C S-GOTOREG
             C S-FREG
             C S-REGCOD
             C S-CODE
             C S-CODOP
             C S-CODELIST
             C S-CREME
             C S-CRELIST
             C S-VALELEM
             C FDFX
             C FD-BT
             C FD-R
             C FD-L
             C FD-S
             C FD-P
             C FD-I
             C FD-E
             C FD-SAUVE
             C FD-REST
  K-SEMAN                          9
             C U-SEMAN
             M DY22
             C U-RANGE
             C K-LEC
             M ERREUR
             M ABANDONLIGNE
             M SUIVAN
             M ERRNONFATAL
  K-ANAL                           19
             C K-LEC
             C K-SEMAN
             M DY22
             M ERRNONFATAL
             M ERREUR
             C K-LIN1
             C U-RANGE
             M ABANDONLIGNE
```

```
          C U-EXECUTION
C ANAL                          3
          C K-ANAL
M NOYEAU                        1
          C ANAL
M NPROJET                       1
M ENTETE                        1
M HP                            1
          I SAUT
          M ENTETE
M BP                            1
          I SAUT
          M ENTETE
M BHP                           2
          M BP
          M HP
M ECRIT                         2
          M BHP
M BL                            2
          M BHP
          I SAUT
M CADRE                         5
          M HP
          M BL
          M BP
M EDLIB                         3
          M ECRIT
M EDLICO                        3
          M ECRIT
M EDTXT                        11
          M ECRIT
M XEDS                          9
          M CADRE
          M HP
          M BP
M PEDS                         16
          M CADRE
          M HP
          M ECRIT
          M BL
          M EDLIB
          M BP
M REDS                         23
          M CADRE
          M HP
          M EDLIB
          M EDTXT
          M BL
          M BP
M DOSANA                        6
          I DATSYS
          M CADRE
M EDETAT                        3
          M ECRIT
C MOTRESERV                     1
** FIN LISTE **
        RUN FN:M
```

C12 - MISE EN OEUVRE ET EVALUATION DE NOYAU

1 - Initialisation

NOYAU se présente comme un ensemble de cartes perforées comprenant :

- une définition de structure, nécessaire à la gestion des transactions et à la documentation (voir § C-10)
- un catalogue de programmes : programmes assurant cette gestion et cette documentation ainsi que les utilitaires : U-SEMAN, U-EXECUTION, U-BGEX, ANAL, U-INIT, U-AMAJ, U-LIB, U-GLED, LEC, NPROJET, DOSANA, SEDT, XEDS, CADRE, ECRIT, BL, BP, HP, EDLICO, EDTXT, GLOSSAIRE (voir § C-11)
- une initialisation du fichier : création du projet NOYAU ; saisie sous le projet NOYAU du schéma des schémas et par l'intermédiaire de celui-ci, des schémas CREEX, CREME, CRELIST (voir § C-8) ; saisie, enfin, d'un certain nombre de messages d'erreurs.

2 - La méthodologie de construction d'un prototype

L'utilisation de NOYAU pour la construction d'un prototype passera alors par les phases suivantes :

- faire l'initialisation (cf. C12-1),
- saisir les schémas de transaction propres au prototype,
- rajouter la structure et cataloguer les différents programmes sémantiques et de traitement,
- modifier et recompiler les programmes U-SEMAN et U-EXECUTION qui réalisent l'appel associatif aux programmes précédemment catalogués,
- saisir les messages d'erreurs associés aux programmes sémantiques ci-dessus,
- cataloguer les différents programmes d'édition correspondant à chaque nature possible d'objet,
- définir les programmes de sélection d'objets à lister,
- définir les dossiers de sortie.

Ecriture de programmes : variables de travail

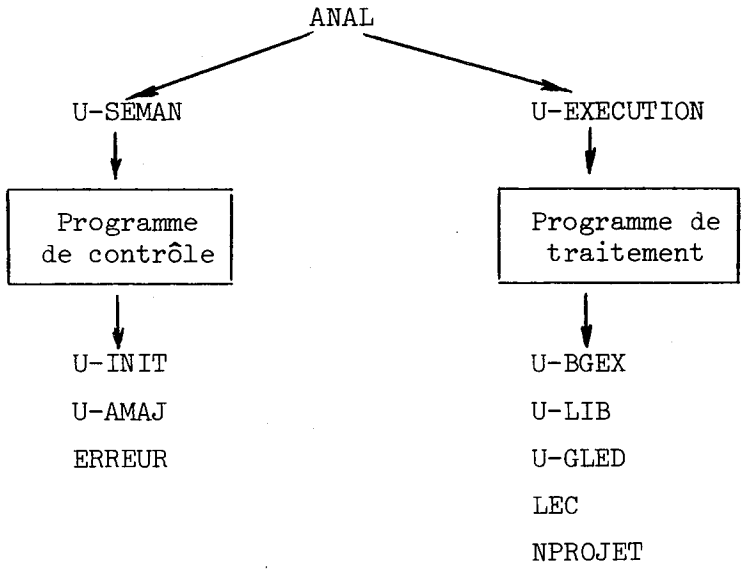
En conversationnel, il est utile que chaque utilisateur ait un jeu de variables qui lui soit personnel : dans l'écriture des programmes, la variable Y22 pourra caractériser un utilisateur on line ; on disposera d'autre part des variables :

- numériques : Syi dy22 (i=1,2,...,6)
- alphanumériques : SZi Dy22 (i=1,2,...,6)

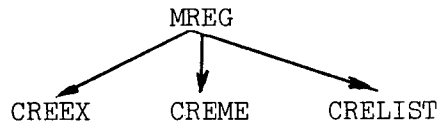
qui seront locales à chaque utilisateur (voir entité DICINIT).

3 - Résumé synthétique

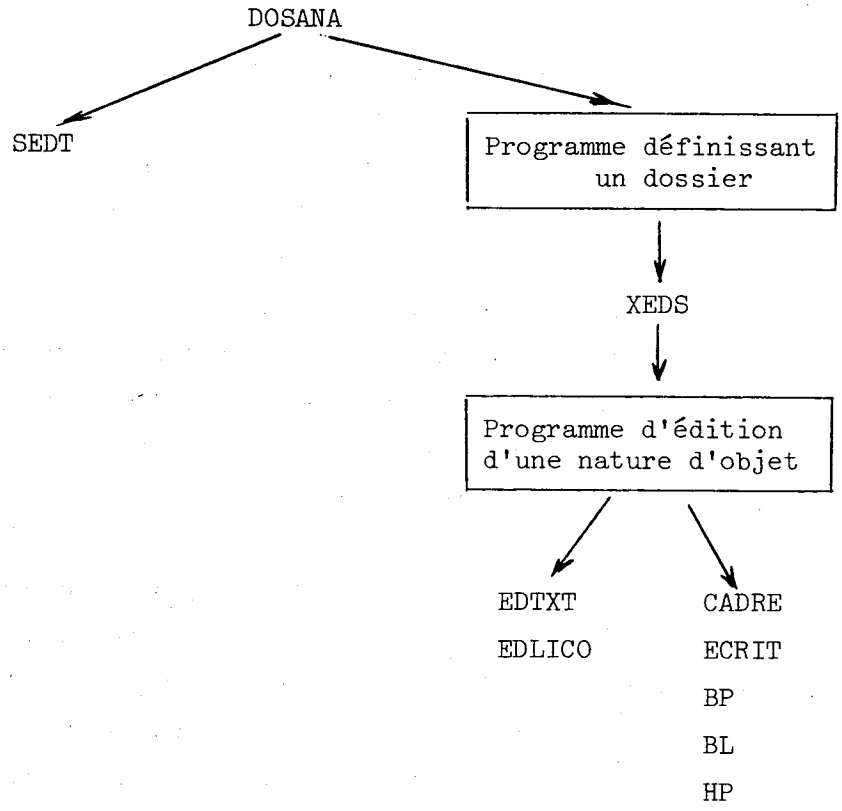
(1) Outils de saisie de l'information (dans l'ordre où ils sont appelés) :



(2) Schémas de transaction :



(3) Outils de restitution de l'information (dans l'ordre où ils sont appelés) :



4 - MEDOC : une application de NOYAU

MEDOC [L14] propose un logiciel permettant de documenter des systèmes d'informations ; comme dans toute application documentaire, on y trouve des programmes de saisie de la documentation ainsi que des programmes de restitution de cette documentation.

La saisie de la documentation en MEDOC se fait par l'intermédiaire de *transactions* soumises à des règles précises, que nous avons appelées : *schémas de transactions* ; la restitution de la documentation se fait, elle, sous forme de *dossiers de sortie*.

- Une première version de MEDOC fut réalisée, indépendamment de NOYAU, correspondant à un volume d'environ 6000 cartes
- NOYAU a ensuite été utilisé (volume d'environ 2500 cartes). Le volume de cartes greffées à NOYAU pour obtenir MEDOC a été alors d'environ 2000 cartes (dont 1000 de programmes et 1000 de données).

Les gains réalisés dans la deuxième implémentation peuvent alors être évalués de la façon suivante :

- gain en volume global, d'environ 1500 cartes
- gain sur le volume de cartes spécifiques à MEDOC : environ 4000 cartes, et donc gain équivalent pour toute autre application qui s'appuyerait sur NOYAU
- enfin, gain appréciable en durée de programmation, puisqu'aux 6000 cartes de programmes de la première version ne correspondent plus que 1000 cartes dans la seconde
- les 1000 cartes de données saisies pour la deuxième version de MEDOC correspondent à la définition des schémas de transactions propres à MEDOC ; sur ce volume, une moitié environ correspond à la documentation de ces schémas : en tout point d'une transaction, que peut-on faire, sous quelles conditions, quelles en seront les conséquences ?
- enfin, une gestion de transactions appuyée sur NOYAU impose une méthodologie dans le découpage des programmes et dans le choix des variables , d'où une plus grande rapidité d'écriture ; sur les 1000 cartes de programmes de la seconde version, 1/3 environ correspond à des programmes de cinq instructions en moyenne ; la maintenance en est rendue plus facile.

BIBLIOGRAPHIE

- OUVRAGES

[L1] J.R. ABRIAL

"Projet SOCRATE".

Cours avancé sur l'architecture des systèmes informatiques - Alpe d'Huez - Décembre 1972.

[L2] M. ADIBA, C. DELOBEL

"Les modèles relationnels de bases de données".

Laboratoire d'informatique - USMG - Grenoble - Avril 1976.

[L3] R. BARBOSA

"Contributions à l'étude sémantique des bases de données ; application au système SOCRATE".

Thèse de docteur-ingénieur - USMG - GRENOBLE - Octobre 1975.

[L4] J.L. CAVARERO

"Modèle d'évaluation de systèmes d'organisation".

Thèse de Docteur-Ingénieur - U.S.M.G. - Grenoble - 1972.

[L5] A. de CHELMINSKI

"Le projet MACSI-1".

Thèse de docteur-ingénieur - USMG - GRENOBLE - Juin 1975.

[L6] Y. CHIARAMELLA

"COBOL" : éléments de programmation structurée".

Département informatique IUT 2 - USSG - GRENOBLE - 1974.

[L7] J. COURTIN, J. VOIRON

"Introduction à l'algorithmique et aux structures de données".

Département informatique IUT 2 - USSG - GRENOBLE - 1974.

[L8] CXP

"Etude comparative des méthodes d'analyse".

Centre d'eXpérimentation des Packages - Etude 008.

[L9] C. DELOBEL

"Contributions théoriques à la conception et l'évaluation d'un système d'informations appliqué à la gestion".
Thèse d'état - USMG - GRENOBLE - Octobre 1973.

[L10] C. DELOBEL

"Les systèmes de bases de données".
Ecole d'été de l'AFCET - Rabat - Juillet 1975.

[L11] JC. DERNIAME

"CIVA : un système de programmation modulaire".
Thèse d'état - NANCY - Janvier 1974.

[L12] JF. DESNOS

"Elaboration automatique d'emplois du temps".
Thèse de docteur-ingénieur - USMG - GRENOBLE - Novembre 1971.

[L13] D. DROMARD

"Représentation d'une procédure de transmissions par des automates d'états finis".
Thèse de docteur-ingénieur - PARIS VI - Juin 1974.

[L14] IFC-GRADIA

"Le projet MEDOC : un prototype de MEthode de DOCumentation".
Rapport technique - GRENOBLE - Mai 1975.
IFC - IUT 2 - Place Doyen Gosse - 38031 GRENOBLE-CEDEX.

[L15] IFC-GRADIA

"BADOI : prototype d'une BAse de DOnnées Interadministrative".
Rapport technique - GRENOBLE - Juin 1975.
IFC - IUT 2 - Place Doyen Gosse - 38031 GRENOBLE-CEDEX.

[L16] COLLOQUE INFORSID

"La recherche en informatique des organisations".
CAEN - Novembre 1976.

[L17] R. LAURINI
"Méthode de construction des bases de données".
Département informatique - INSA - VILLEURBANNE - 1975-76.

[L18] M. LEONARD
"Aides algorithmiques à la conception de bases de données"
Thèse de docteur-ingénieur - USMG - GRENOBLE - Juin 1976.

[L19] M. LUCAS, PM. SCHOLL
"Proposition pour une initiation à l'algorithmique".
Laboratoire d'informatique - USMG - GRENOBLE - Décembre 1975.

[L20] J. MELESE
"La gestion par les systèmes - Essai de praxéologie".
Ed. Hommes et Techniques. - 1968.

[L21] F. PECCOUD
"Réflexions sur la construction d'un modèle d'aide à la conception des systèmes informatiques de gestion".
Ecole d'été de l'AFCET - ALES - Juillet 1971.

[L22] F. PECCOUD
"MACSI = Méthodes d'Aide à la Conception des Systèmes d'Informations"
Thèse d'état - USMG - GRENOBLE - Juin 1975.

[L23] SOCRATE SIRIS2/SIRIS3
"Manuel d'utilisation - Manuel d'opération".
Brochures CII - Juillet 1973.

[L24] ANSI/SPARC
"Report Study Group on Data Base Management Systems".
Bulletin of ACM-SIGMOD, vol. 7, number 2 - Février 1975.

[L25] H. TARDIEU, H. HECKENROTH, D. NANCI
"Etude d'une méthodologie d'analyse et de conception d'une base de données : le schéma de référence".
CETE - AIX-EN-PROVENCE - Mai 1975.

[L26] J. THULY, A. SAUNIER
"Objectifs et conception d'une base de données".
Tomes 1 et 2 - Editions d'informatique - PARIS - 1976.

[L27] J. TRICOT
"Guide pratique des bases de données".
Editions d'informatique - PARIS - 1976.

[L28] JG. VAUCHER
"La programmation avec SIMULA-67".
Université de Montréal - Juillet 1973.

[L29] B. VAUQUOIS
"Calculabilité des langages".
C3 - maîtrise d'informatique - USMG - GRENOBLE - 1970.

- ARTICLES

[A1] J.R. ABRIAL

"Data semantics".

IFIP Working conference, Cargese - CORSE - Avril 1974.

[A2] M. ADIBA, C. DELOBEL, M. LEONARD

"A unified approch for modelling data in logical data base design".

IFIP - TC2 - Working conference - Freudenstadt - Janvier 1976.

[A3] M. ADIBA, M. LEONARD

"TS2 : Présentation d'une structure CODASYL en termes de structure relationnelle".

Note technique - laboratoire d'informatique - USMG - GRENOBLE - Juin 1975.

[A4] C. BACHMAN

"The programmer as Navigator".

CACM - vol. 16, n° 11 - Novembre 1973.

[A5] P. BERGER

"Les maquettes de systèmes de gestion ; une idée prometteuse, mais encore jeune".

Informatique et gestion - n° 53 - Janvier 1974.

[A6] C. COCHET

"Les conditions de la maîtrise d'un projet en informatique de gestion".

Informatique et gestion - n° 76 - Mars 1976.

[A7] CODASYL

"Data Base Task Group Report".

ACM - NEW-YORK - Avril 1971.

[A8] CODASYL
"Data Description Language".
Handbook 112, US. Departement of commerce - Janvier 1974.

[A9] E.F. CODD
"A relational model of data for large shared data banks".
CACM - vol. 13, n° 6 - Juin 1970.

[A10] J.D. COUGER
"Evolution of business system analysis techniques".
ACM computing surveys - vol. 5, n° 3 - Septembre 1973.

[A11] M. DAMBRINE
"L'organisation des données d'un système".
Séminaire INFORSID II - PONT-A-MOUSSON - Mai 1976.

[A12] M. DAPPE, JF. ROCH
"Le projet BADOI : exemple de réalisation".
Informatique et gestion - n° 85 - Mars 1977.

[A13] I. DARMON
"Base de données, maquette et dialogue avec les utilisateurs".
Informatique et gestion - n° 85 - Mars 1977.

[A14] EDP ANALYSER
"The current status of data management".
Février 1974.

[A15] J.J. FLORENTIN
"Information reference coding".
CACM - vol. 19, n° 1 - Janvier 1976.

[A16] R. GERRITSEN
"A preliminary system for the design of DBTG data structure".
CACM - vol. 18, n° 10 - Octobre 1975.

[A17] JP. GIRAUDIN, M. LARCHER, F. PECCOUD
"Le projet MACSI".
Séminaire de programmation - USMG - GRENOBLE - Février 1974.

[A18] JP. GIRAUDIN, F. PECCOUD
"Le projet MACSI-P : son modèle de représentation des données et des traitements".
Séminaire INFORSID-2 - St-PIERRE-DE-CHARTREUSE - Octobre 1976.

[A19] JP. GIRAUDIN, D. JAHU, F. PECCOUD
"Prototypes d'applications informatiques de gestion : opportunité et conditions de faisabilité et d'utilisation".
AFCET/TTI 1976 - GIF SUR YVETTE - Novembre 1976.

[A20] JP. GIRAUDIN, F. PECCOUD
"Réflexions sur la construction d'un prototype pour faciliter la conception d'une application informatique".
Informatique et gestion - n° 85 - Mars 1977.

[A21] JC. HUMBLOT
"La modélisation appliquée à la conception de systèmes informatiques".
Informatique et gestion - n° 74 - Janvier 1976.

[A22] F. LAPADU - HARGUES
"Utilisation d'un automate pour des applications conversationnelles".
INTER APL SLIGOS - n° 7 - Juillet 1975.

[A23] C. ROLLAND
"Un système de pilotage de la conception des systèmes d'information".
Informatique et gestion - n° 85 - Mars 1977.

[A24] D. TEICHROEW
"A survey of languages for stating requirements for computer-based information systems".
Fall joint computer conference - 1972.

[A25] D. TEICHROEW, W.J. RATAJ , E.A. HERSHEY
"An introduction to computer-aided documentation of user requirements for computer based information processing systems".
ISDOS Working paper - n° 72 - Mars 1973.

[A26] D. TEICHROEW, E.A. HERSHEY, M.J. BASTARACHE
"An introduction to PSL/PSA".
ISDOS Working paper - n° 86 - Mars 1974.

[A27] S.J. WATERS
"Méthodologie assistée par ordinateur dans la conception des systèmes informatiques".
L'informatique nouvelle - n° 69 - Janvier 1976.

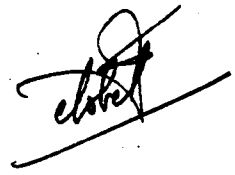
Dernière page d'une thèse

VU

Grenoble, le 23 Mai 1977

Le Président de la thèse

C. DELOBEL



Vu, et permis d'imprimer,

Grenoble, le

Le Président de l'Université
Scientifique et Médicale