M.A.C.S.I : méthode d'aide à la conception des systèmes d'informations

François Peccoud

THESE présentée à

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

POUR OBTENIR LE GRADE DE
DOCTEUR ES-SCIENCES

François PECCOUD

M.A.C.S.I.

Méthode d'Aide à la Conception des Systèmes d'Informations

Thèse soutenue le 5 juin 1975 devant la commission d'examen

Président : J. KUNTZMANN
Rapporteur : J.C. BOUSSARD
Examinateurs : L. BOLLIET, C. DELOBEL
Invité : Monsieur PAUL

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE		INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE
M. Michel SOUTIF	Présidents	M. Louis NEEL
M. Gabriel CAU	Vice-Présidents	MM. Lucien BONNETAIN Jean BENOIT

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

PROFESSEURS TITULAIRES

MM.	ANGLES D'AURIAC Paul	Mécanique des fluides
	ARNAUD Paul	Chimie
	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
	BERNARD Alain	Mathématiques Pures
Mme	BERTRANDIAS Françoise	Mathématiques Pures
MM.	BEZES Henri	Pathologie chirurgicale
	BLAMBERT Maurice	Mathématiques Pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Georges	Electrotechnique
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET-EYMARD Joseph	Pathologie médicale
	BOUCHERLE André	Chimie et toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BOUSSARD Jean-Claude	Mathématiques appliquées
	BRAVARD Yves	Géographie
	CABANEL Guy	Clinique rhumatologique et hydrologie
	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	Thérapeutique (Neurologie)
	CHIBON Pierre	Biologie animale
	COEUR André	Pharmacie chimique et chimie analytique
	CONTAMIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie pathologique
	CRAYA Antoine	Mécanique
Mme	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBERMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DELORMAS Pierre	Pneumo-Phtisiologie
	DEPORTES Charles	Chimie minérale
	DESRE Pierre	Métallurgie
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée

MM.	DOLIQUE Jean-Michel	Physique des plasmas
	DREYFUS Bernard	Thermodynamique
	DRUCROS Pierre	Cristallographie
	DUGOIS Pierre	Clinique de dermatologie et syphiligraphie
	FAU René	Clinique neuro-psychiatrique
	GAGNAIRE Didier	Chimie physique
	GALLISSOT François	Mathématiques pures
	GALVANI Octave	Mathématiques pures
	GASTINEL Noël	Mathématiques appliquées
	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
	LAJZEROWICZ Joseph	Physique
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie pharmaceutique
	LAURENT Pierre-Jean	Mathématiques appliquées
	LEDRU Jean	Clinique médicale B
	LLIBOUTRY Louis	Géophysique
	LONGEQUEUE Jean-Pierre	Physique nucléaire
	LOUP Jean	Géographie
Mle	LUTZ Elisabeth	Mathématiques pures
	MALGRANGE Bernard	Mathématiques pures
	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Seméiologie médicale
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et pétrographie
	MICOUD Max	Clinique maladies infectieuses
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	MULLER Jean-Michel	Thérapeutique (néphrologie)
	NEEL Louis	Physique du solide
	OZENDA Paul	Botanique
	PAYAN Jean-Jacques	Mathématiques pures
	PEBAY-PEYROULA Jean-Claude	Physique
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	RINALDI Renaud	Physique
	DE ROUGEMONT Jacques	Neuro-chirurgie
	SEIGNEURIN Raymond	Microbiologie et hygiène
	SENGEL Philippe	Zoologie
	SIBILLE Robert	Construction mécanique
	SOUTIF Michel	Physique générale
	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	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
	YOCCOZ Jean	Physique nucléaire théorique

PROFESSEURS ASSOCIES

MM.	CHEEKE John	Thermodynamique
	COPPENS Philip	Physique
	CORCOS Gilles	Mécanique
	CRABBE Pierre	CERMO
	GILLESPIE John	I.S.N.
	ROCKAFELLAR Ralph	Mathématiques appliquées

PROFESSEURS SANS CHAIRE

Mlle	AGNIUS-DELORD Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	AMBROISE-THOMAS Pierre	Parasitologie
	BELORIZKY Elie	Physique
	BENZAKEN Claude	Mathématiques appliquées
	BERTRANDIAS Jean-Paul	Mathématiques pures
	BIAREZ Jean-Pierre	Mécanique
	BILLET Jean	Géographie
Mme	BONNIER Jane	Chimie générale
MM.	BOUCHET Yves	Anatomie
	BRUGEL Lucien	Energétique
	CONTE René	Physique
	DEPASSEL Roger	Mécanique des fluides
	GAUTHIER Yves	Sciences biologiques
	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 Méd. Préventive
	IDELMAN Simon	Physiologie animale
	JOLY Jean-René	Mathématiques pures
	JULLIEN Pierre	Mathématiques appliquées
Mme	KAHANE Josette	Physique
MM.	KUHN Gérard	Physique
	LOISEAUX Jean	Physique nucléaire
	LUU-DUC-Cuong	Chimie organique
	MAYNARD Roger	Physique du solide
	PELMONT Jean	Biochimie
	PERRIAUX Jean-Jacques	Géologie et minéralogie
	PFISTER Jean-Claude	Physique du solide
Mlle	PIERY Yvette	Physiologie animale
MM.	RAYNAUD Hervé	Mathématiques appliquées
	REBECQ Jacques	Biologie (CUS)
	REVOL Michel	Urologie
	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.	VIALON Pierre	Géologie
	VAN CUTSEM Bernard	Mathématiques appliquées

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

MM.	AMBLARD Pierre	Dermatologie
	ARMAND Gilbert	Géographie
	ARMAND Yves	Chimie
	BARGE Michel	Neurochirurgie
	BEGUIN Claude	Chimie organique
Mme	BERIEL Hélène	Pharmacodynamique
M.	BOUCHARLAT Jacques	Psychiatrie adultes
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BRODEAU François	Mathématiques (IUT B)
	BUISSON Roger	Physique
	BUTEL Jean	Orthopédie
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse
	CHARDON Michel	Géographie
	CHERADAME Hervé	Chimie papetière
	CHIAVERINA Jean	Biologie appliquée (EFP)
	COHEN-ADDAD Jean-Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie médicale
	CORDONNIER Daniel	Néphrologie
	COULOMB Max	Radiologie
	CROUZET Guy	Radiologie
	CYROT Michel	Physique du solide
	DELOBEL Claude	M.I.A.G.
	DENIS Bernard	Cardiologie
	DOUCE Roland	Physiologie végétale
	DUSSAUD René	Mathématiques (CUS)
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	FONTAINE Jean-Marc	Mathématiques pures
	GAUTIER Robert	Chirurgie générale
	GENSAC Pierre	Botanique
	GIDON Maurice	Géologie
	GRIFFITHS Michaël	Mathématiques appliquées
	GROS Yves	Physique (stag.)
	GUITTON Jacques	Chimie
	HICTER Pierre	Chimie
	IVANES Marcel	Electricité
	JALBERT Pierre	Histologie
	KOLODIE Lucien	Hématologie
	KRAKOWIAK Sacha	Mathématiques appliquées
Mme	LAJZEROWICZ Jeannine	Physique
MM.	LEROY Philippe	Mathématiques
	MACHE Régis	Physiologie végétale
	MAGNIN Robert	Hygiène et médecine préventive
	MARECHAL Jean	Mécanique
	MARTIN-BOUYER Michel	Chimie (CUS)
	MICHOULIER Jean	Physique (IUT A)
Mme	MINIER Colette	Physique
MM.	NEGRE Robert	Mécanique
	NEMOZ Alain	Thermodynamique
	PARAMELLE Bernard	Pneumologie
	PECCOUD François	Analyse (IUT B)
	PEFFEN René	Métallurgie
	PERRET Jean	Neurologie
	PHELIP Xavier	Rhumatologie
	RACHAIL Michel	Médecine interne
	RACINET Claude	Gynécologie et obstétrique
	RAMBAUD Pierre	Pédiatrie
Mme	RENAUDET Jacqueline	Bactériologie
MM.	ROBERT Jean-Bernard	Chimie-Physique

"MEMBRES DU CORPS ENSEIGNANT DE L'I.N.P.G."

PROFESSEURS TITULAIRES

MM.	BENOIT Jean	Radioélectricité
	BESSON Jean	ELectrochimie
	BONNETAIN Lucien	Chimie Minérale
	BONNIER Etienne	Electrochimie, Electrométallurgie
	BRISSONNEAU Pierre	Physique du solide
	BUYLE-BODIN Maurice	ELectronique
	COUMES André	Radioélectricité
	FELICI Noël	Electrostatique
	PAUTHENET René	Physique du solide
	PERRET René	Servomécanismes
	SANTON Lucien	Mécanique
	SILBER Robert	Mécanique des fluides

PROFESSEUR ASSOCIE

M.	BOUDOURIS Georges	Radioélectricité

PROFESSEURS SANS CHAIRE

MM.	BLIMAN Samuel	Electronique
	BLOCH Daniel	Physique du solide et cristallographie
	COHEN Joseph	Electrotechnique
	DURAND Françis	Metallurgie
	MOREAU René	Mécanique
	POLOUJADOFF Michel	ELectrotechnique
	VEILLON Gérard	Informatique fondamentale et appliquée
	ZADWORNY François	Electronique

MAITRES DE CONFERENCES

MM.	BOUVARD Maurice	Génie mécanique
	CHARTIER Germain	Electronique
	FOULARD Claude	Automatique
	GUYOT Pierre	Chimie minérale
	JOUBERT Jean Claude	Physique du solide
	LACOUME Jean Louis	Géophisique
	LANCIA Roland	Physique atomique
	LESPINARD Georges	Mécanique
	MORET Roger	Electrotechnique nucléaire
	ROBERT François	Analyse numérique
	SABONNADIERE Jean Claude	Informatique fondamentale et appliquée
Mme	SAUCIER Gabrièle	Informatique fondamentale et appliquée

MAITRE DE CONFERENCES ASSOCIE

M.	LANDAU Ioan Doré	Automatique

CHARGE DE FONCTIONS DE MAITRES DE CONFERENCES

M.	ANCEAU François	Mathématiques appliquées

Fait à Saint Martin d'Hères JANVIER 1974

MM.	ROMIER Guy	Mathématiques (IUT B)
	SHOM Jean-Claude	Chimie générale
	STIEGLITZ Paul	Anesthésiologie
	STOEBNER Pierre	Anatomie pathologique
	VROUSOS Constantin	Radiologie

MAITRES DE CONFERENCES ASSOCIES

MM.	COLE Antony	Sciences nucléaires
	FORELL César	Mécanique
	MOORSANI Kishin	Physique

CHARGES DE FONCTIONS DE MAITRES DE CONFERENCES

MM.	BOST Michel	Pédiatrie
	CONTAMIN Charles	Chirurgie thoracique et cardio-vasculaire
	FAURE Gilbert	Urologie
	MALLION Jean-Michel	Médecine du travail
	ROCHAT Jacques	Hygiène et hydrologie

Fait à Saint Martin d'Hères, OCTOBRE 1974.

à mes parents,

Qu'il me soit permis de remercier

Monsieur le Professeur KUNTZMANN qui m'a fait l'honneur de présider ce jury après m'avoir, pendant tant d'années, offert toute sa confiance ;

Monsieur le Professeur BOLLIET et Monsieur le Professeur BOUSSARD avec lesquels j'ai pu travailler jusqu'ici dans une collaboration amicale que je souhaite poursuivre ;

Monsieur DELOBEL, Maître de Conférences à l'Université Scientifique et Médicale de Grenoble, qui m'a initié aux problèmes posés par l'Informatique de Gestion et m'a ensuite constamment aidé de ses conseils et de son expérience ;

Monsieur PAUL, chargé de mission à la Direction de la Prévision (Ministère des Finances) et, avec lui, toute l'équipe du GRADIA (1) dont les choix dialectiques entre recherche et problèmes quotidiens m'ont offert la chance irremplaçable de confronter mes propositions aux exigences de la réalité.

J'exprime aussi très volontiers ma gratitude à tous ceux par qui le travail, présenté ici, a été possible.

Avec Madame ZIDANI et Messieurs ARCHER, BERGONIER, CAVARERO, GIRES et GOUNOT, j'ai appris à poser des problèmes sans toujours, hélas, savoir les résoudre.

Avec Messieurs BLANGERO, de CHELMINSKI, GIRAUDIN, JAHU, LARCHER, PERNET, ROCH et SOLDANO, j'ai compris le difficile équilibre qui doit être respecté entre l'ivresse des idées et les contraintes de la réalisation.

Madame DOREL, enfin, a fait preuve, pour la frappe de cet ouvrage, d'un goût du travail bien fait dont je lui suis très reconnaissant.

Mais je me dois de dire surtout que cette recherche fut aussi une épreuve morale : sans le soutien de mon frère et la patience bienveillante et sans limites de Sophie, ma femme, je n'aurais sans doute pas pu la surmonter.

(1) - GRADIA : Groupe de Recherche et d'Appui pour le Développement de l'Informatique dans l'Administration - Ministère de la Coopération.

SOMMAIRE

SOMMAIRE

CHAPITRE 1

AVANT-PROPOS

1.1. INTRODUCTION

Les propositions présentées dans cet ouvrage ont pour domaine d'application l'informatique appliquée à la gestion.

Mais qu'est-ce que l'informatique appliquée à la gestion, l'informatique de gestion ?

Si la question vaut d'être posée, puisque récemment encore un colloque avait lieu pour l'examiner [24], notre propos n'est pas ici de donner une réponse mais simplement de rappeler trois constatations de fait :

1 - 80 % des ordinateurs sont affectés à des tâches de gestion.

2 - Un certain nombre d'entre eux sont suffisamment mal utilisés au point que des sociétés de service ont jugé utile, depuis quelques années, d'orienter leurs compétences pour être capables de porter un diagnostic sur les déficiences constatées : elles proposent un "AUDIT INFORMATIQUE" [14].

3 - Dans la panoplie des améliorations qu'elles conseillent figurent en bonne place des METHODES D'ANALYSE. Plus d'une vingtaine sont actuellement proposées sur le marché. Le projet MACSI procède, au niveau de la recherche, des mêmes intentions que celles ayant conduit à la fabrication et la diffusion de ces produits méthodologiques.

1.2. RAPPELS

Sous le terme de METHODE D'ANALYSE sont regroupés en fait des moyens très différents : leur seul point commun se situe au niveau des finalités qui leur sont associées.

Il est possible de dire de façon très ambiguë que toute méthode d'analyse se propose d'aider les responsables d'un projet informatique à résoudre certaines des difficultés rencontrées dans la conception, la réalisation et le contrôle d'un projet informatique lorsque celui-ci concerne la construction d'un système d'information adapté à des tâches de gestion.

Identiques quant à leurs objectifs, ces méthodes diffèrent dans leur ccntenu car leurs auteurs n'ont pas toujours le même point de vue sur ce que sont ces difficultés ou sur les solutions pour y remédier.

Pour le lecteur qui voudrait examiner avec soin ce que ces différentes méthodes contiennent, signalons qu'il existe quelques études analytiques de bonne qualité qui les comparent :

- Un numéro spécial d'Informatique et Gestion [23] fait une présentation rapide de quelques produits assez représentatifs de ceux qui sont utilisés en France. Avec une étude plus récente du CXP, il est possible de trouver un examen beaucoup plus approfondi des principales méthodes actuellement employées [11].
- Deux études de Computing Surveys [4] [9] proposent, au-delà des méthodes, une classification des problèmes rencontrés dans un projet informatique de gestion et des types de solutions proposées. L'étude du Professeur D. TEICHROEW [39] fait un inventaire des "langages d'analyse" existant en 1972.
- L'ouvrage du BIFOA [28] fait une présentation et une évaluation précise de six méthodes d'analyse essistée par ordinateur que sont les projets AUTOSATE, CADIS, CASCADE, DATAFLOW, SYSTEMATICS et ISDOS.
- Enfin, avec les références [40] et [41] sont précisément définis les objectifs et spécifications externes du projet ISDOS qui est sans doute un des plus développés dans le domaine. On pourra en trouver un bref descriptif en français dans [38].

Le lecteur n'aura peut-être pas le temps de prendre connaissance de toutes ces études comparatives. Nous allons alors essayer d'en dégager les aspects essentiels en prenant soin d'abord de rappeler quelques éléments de terminologie et ensuite d'esquisser, en termes de services rendus, les fonctions assurées par la plupart des méthodes d'analyses.

1.2.1. Les phases d'un projet informatique

En effet, il faut d'abord se mettre d'accord sur quelques termes constamment employés par la suite et, en particulier, ceux qui dénotent les différentes étapes de réalisation d'un projet informatique. Nous avons reporté sur le tableau de la figure 1 les noms donnés aux différentes phases d'un projet selon différents auteurs.

Pour MACSI, nous distinguerons les étapes suivantes :

- L'ETUDE D'OPPORTUNITE doit préciser le domaine de gestion concerné et classer les principaux objectifs que l'on assigne au produit informatique au moment du démarrage de son étude.

- Pendant l'ANALYSE FONCTIONNELLE, sont définies les fonctions de traitement de l'information et d'organisation, que doit offrir le système d'information étudié pour satisfaire aux objectifs.

- L'ANALYSE ORGANIQUE est la phase durant laquelle doivent être choisis puis totalement décrits les meilleurs éléments techniques qui permettent de réaliser les fonctions préalablement définies. Cette étape de choix des moyens techniques doit tenir compte de contraintes comme le coût du projet, des délais de réalisation impératifs, des normes de sécurité de fonctionnement, le matériel et les logiciels disponibles.

- La PROGRAMMATION et les TESTS des programmes sont, dans la phase de réalisation, les tâches les plus complexes et les plus coûteuses. Tout doit être entrepris pour en améliorer la qualité.

- Le LANCEMENT de l'application exige des travaux de constitution de fichiers et de formation du personnel d'exécution.

- La MAINTENANCE d'une application informatique qui doit être assurée après chaque demande de modification ou d'amélioration formulée par ses utilisateurs. Elle est l'occasion de bien de difficultés lorsque la documentation fait défaut ou que les programmes sont mal construits.

Référence bibliographique	ETAPES DE REALISATION D'UN PROJET INFORMATIQUE							
PROTEE [11]	Diagnostic	Analyse préalable	Emploi	Organisation des chaînes de traitements	Organisation des programmes	Ecriture et mise au point	Lancement	Maintenance
ISDOS [40]	Perception of need	Logical system design		Physical system design	Construction	Testing, Conversion and installation	Opération	Maintenance
COUGER [9]	Documentation of the existing system	Logical design		Physical design	Programming and Procedures développement	Implémentation		
CXP [11]	ANALYSE GLOBALE	ANALYSE de CONCEPTION		ANALYSE ORGANIQUE				Maintenance
MACSI	ETUDE D'OPPORTUNITE	ANALYSE FONCTIONNELLE		ANALYSE ORGANIQUE	PROGRAMMATION et TESTS		Lancement	Maintenance

date de début du projet

Date d'utilisation de l'application

Temps

FIGURE 1

1.2.2. Caractéristiques générales des méthodes d'analyse assistées par ordinateur

Si, en informatique, la multitude de langages de programmation n'a pas empêché la création de l'algorithmique, de même en informatique de gestion l'examen de plusieurs méthodes d'analyse, limité ici aux seules méthodes assistées par ordinateur, permet de dégager quelles sont, au-delà des variantes, les fonctions de bases qu'elles assurent.

Nous souhaitons dégager très succinctement ces fonctions pour pouvoir mieux situer les objectifs du projet MACSI par rapport aux produits existants.

Pour évaluer une méthode d'analyse, il faut répondre aux trois questions que suggère la Figure 2 (page 1.7).

- Quel est le modèle de représentation d'un système d'information qu'elle propose ?
- Quelles sont les fonctions des programmes d'aide à l'analyse disponibles ?
- Quelle pédagogie est associée à la méthode pour permettre de la maîtriser ?

A - Le modèle de description d'un projet informatique est la présentation des concepts fondamentaux qui doivent être utilisés pour spécifier sans ambiguïté une application. Les règles de description utilisant ces concepts doivent être précisées dans un manuel de référence pour que tous les membres d'une équipe utilisant la méthode produisent une documentation standardisée. Dans cette perspective de standardisation, on associe aux règles de description des formulaires préimprimés de description. Une bonne connaissance du modèle de description permet de savoir s'il est adéquat, ou non, pour décrire un projet déterminé.

B - Les programmes d'aide à l'analyse exploitent un fichier documentaire où sont stockées, à partir de formulaires standards, les spécifications d'un projet informatique.

Selon leur fonction, ces programmes peuvent appartenir à l'un des trois groupes suivants :

P1 - Programmes de gestion de la documentation qui assurent soit la mise à jour du fichier documentaire, soit l'exploitation du contenu du fichier pour fournir une documentation sélective facilitant une prise de connaissance ou une diffusion des spécifications du projet.

P2 - Programmes assurant des contrôles automatiques de cohérence du projet. Ils dépendent essentiellement de la structure du modèle de description retenu. Ils permettent d'alléger le travail des responsables du projet en vérifiant de façon automatique que la documentation est cohérente avec certaines normes de description ou certains critères de qualité.

P3 - Programmes d'aide à la programmation. Il s'agit essentiellement de traducteurs qui, à partir des spécifications des données, des fichiers et de la logique des traitements, génèrent tout ou partie du source des programmes de l'application. Le plus souvent le source de ces programmes est en COBOL. L'intérêt de ces traducteurs est de raccourcir la phase de programmation du projet.

C - Une pédagogie conçue pour apprendre aux analystes, qui auront à se servir de la méthode, la définition de ses concepts de base, et l'utilisation efficace des moyens (formulaires et programmes) qu'elle offre.

Précisons quel peut être l'intérêt d'utiliser des programmes de documentation automatique ou de contrôle de cohérence.

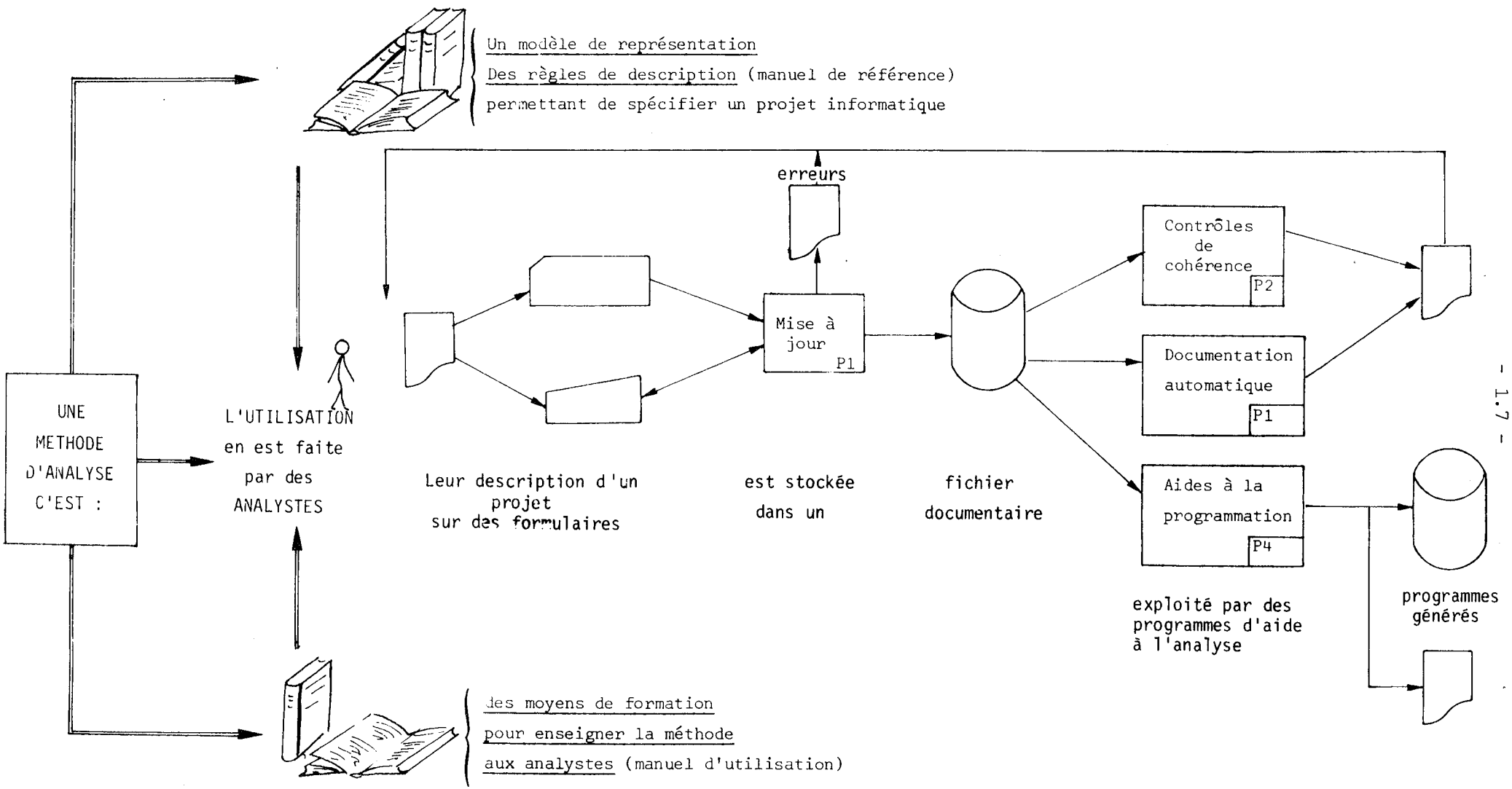

Un modèle de représentation
Des règles de description (manuel de référence)
permettant de spécifier un projet informatique
erreurs
Contrôles de cohérence
P2
Mise à jour
P1
Documentation automatique
P1
Aides à la programmation
P4
UNE METHODE D'ANALYSE C'EST :
L'UTILISATION en est faite par des ANALYSTES
Leur description d'un projet sur des formulaires
est stockée dans un
fichier documentaire
exploité par des programmes d'aide à l'analyse
programmes générés
des moyens de formation
pour enseigner la méthode
aux analystes (manuel d'utilisation)

FIGURE 2

- Objectifs des programmes de documentation automatique

Il faut, en effet, se poser la question de savoir pourquoi une simple documentation, gérée manuellement sous forme d'un dossier d'analyse, n'est pas éventuellement suffisante.

L'expérience pédagogique de suivi de certains petits projets d'élèves, comme la participation à certains projets importants groupant une équipe d'au moins 5 personnes, suggèrent les remarques suivantes :

= Dans de gros projets, le volume de la documentation est tel (plusieurs mètres d'épaisseur de papier pour certains) que sa mise à jour est très difficile et sa consultation quelquefois malaisée. Pour ces gros projets, le volume de cette documentation justifie à lui tout seul la consultation et la mise à jour de fichiers documentaires gérés automatiquement.

= La qualité d'une documentation d'être à jour est souvent absente de projets où l'on dispose uniquement des spécifications initiales qui ne sont pas mises à jour au fur et à mesure des modifications du projet. La mise à jour d'une documentation est fondamentalement jugée comme une tâche ingrate par la plupart des techniciens : elle est assurée correctement uniquement si la direction technique l'impose, si les participants au projet sont convaincus de son intérêt, et, enfin, si les moyens de la réaliser la facilitent au maximum.

Il faut bien constater que la modification d'une documentation manuscrite n'est pas simple car il faut souvent recopier beaucoup plus de documentation que les seules informations qui sont modifiées ou ajoutées. Cette difficulté amène quelquefois les analystes à produire la documentation a posteriori une fois la programmation et les tests terminés. Par contre, si la documentation est tenue à jour sur fichier magnétique, les seules modifications introduites par un éditeur de textes peuvent être répercutées dans toutes leurs conséquences vers les différentes personnes intéressées. C'est ainsi que pendant deux ans, à l'IMAG, le manuel utilisateur du produit Socrate, stocké sur un fichier CMS, était modifié sans difficulté avec l'éditeur CMS : tout nouvel utilisateur de Socrate pouvait demander l'édition de la dernière version. Ensuite, la version 4 de ce logiciel a fait l'objet d'une brochure [37] en Décembre 1973 à laquelle une première annexe a été ajoutée en Avril 1974 : avec cette seconde solution, on a constaté que

les modifications mineures apportées au produit sont transmises aux utilisateurs dans des délais plus longs.

- Objectifs des programmes d'aide au diagnostic

Le rôle d'un chef de projet est toujours de diagnostiquer, à partir des spécifications d'un projet, pour quelles raisons celui-ci pourrait être un échec s'il était réalisé conformément à ces spécifications. Ce travail, long et difficile, peut être facilité si certains contrôles fastidieux sont automatisés et il peut même gagner en qualité car certaines tâches de vérification sont impossibles sans moyens de calcul. Ainsi, les contrôles suivants sont difficiles à réaliser manuellement :

= Contrôle systématique du respect des règles de nomenclatures : il est très difficile, même dans un petit projet d'élève, de vérifier que les règles de codification sont respectées et surtout que dans la liste des éléments spécifiés (ensemble des données, des fichiers ou des programmes...) chacun a bien un code discriminant qui le distingue de tous les autres.

= Contrôle de présence obligatoire de certaines spécifications pour certains éléments. Souvent, en effet, un projet est médiocre beaucoup moins par les spécifications existantes que par l'absence de certaines informations cependant indispensables. Il est très facile par des programmes de contrôle de vérifier que, dans un certain état d'un projet, toutes les valeurs de certaines informations obligatoires existent réellement dans le fichier documentaire.

= Certains contrôles de cohérence du projet, fondés sur une exploitation de nature combinatoire des graphes correspondant à des relations de correspondances entre différents constituants du projet (correspondance par exemple entre données et programmes, données et fichiers, etc ...), ne peuvent se gérer manuellement qu'avec des matrices booléennes. Celles-ci sont inexploitables dès qu'elles atteignent une taille supérieure à 20 lignes ou colonnes. Par contre, la manipulation de ces relations par programmes gérant des pointeurs ou des fichiers inverses est relativement aisée pour autant qu'est utilisé un software facilitant ce genre d'algorithmes.

1.2.3. Les causes d'une multiplicité des méthodes d'analyse

Les nombreuses méthodes disponibles aujourd'hui, si elles répondent toutes au schéma fonctionnel précédent, n'en sont pas moins très différentes les unes des autres. Cette diversité a pour origine les différences d'approche possibles dans la manière de conduire l'analyse selon les projets pris en charge. Plusieurs facteurs doivent être précisés, dès l'étude d'opportunité d'un projet, qui orienteront la méthode retenue pour conduire ce projet. Il peut être utile de les rappeler ici.

A - Le vocabulaire technique, utilisé dans les centres informatiques, n'est pas le même selon les centres.

De nombreux vocables ont des acceptions différentes suivant les centres où ils sont employés, ou bien, le même concept peut être approximativement évoqué en faisant usage de mots différents.

- On parle ainsi indifféremment d'OS, d'operating system ou de système d'exploitation.
- Avec beaucoup d'ambiguïté selon les cas, on discute à propos de "tâche", de "traitement", de "procédure automatisée", de "job", de "programme", d'"UT", d'"unité de traitement".
- On évoque pêle-mêle des "segments", des "blocs", des "steps", voire des "modules".
- Certains utilisent le terme "fichier de gestion", là où d'autres désignent des "fichiers mouvements", etc ... etc ....

L'énumération de ces quelques exemples souligne que les différences de vocabulaire, d'un centre informatique à l'autre, sont dues, pour une part, à l'introduction du jargon franglais, propre à chaque constructeur au moment de l'équipement de chaque centre et, pour une autre part, au manque actuel de standardisation des concepts informatiques constaté malgré les efforts importants de l'AFNOR ou du groupe CODASYL.

Les différentes méthodes d'analyse proposées ne présentent pas non plus de standardisation du vocabulaire. Ainsi [11] :

- MINOS parle de "MOT", ARMIN-PARM ou CORIG de "RUBRIQUE"
- MINOS ou ARIANE parlent de "MODULES", CORIG de "SEGMENT" et de "FONCTION".

B - Les objectifs d'un projet définissent les critères suivant lesquels le projet sera jugé. Selon les cas, ces critères sont différents.
Tel projet sera jugé essentiellement sur la performance à l'exécution des programmes, tel autre sur la rapidité de la mise en service, quitte à sacrifier la vitesse des programmes en exploitation, tel autre enfin sur les possibilités de modification immédiate du code écrit. Selon les cas, les méthodes de travail et les moyens employés ne seront pas les mêmes. Dans le premier cas, une attention particulière pour les jeux d'essais et des commentaires précis sur les programmes écrits en assembleur s'imposent. Pour le second type de projet, il faut tenir à jour avec précision une documentation sur l'avancement du projet (PERT, fiches d'attachements) permettant d'évaluer en permanence les conséquences de tout retard observé ou prévisible. Dans le troisième cas, une conception rigoureusement modulaire de l'application est une nécessité. Avec, comme support, le livre de WALSH [43] qui distingue au moins une douzaine de guides documentaires différents suivant la nature du projet, on peut illustrer plus précisément la dépendance de la documentation vis à vis de la spécificité du projet.

C - Les contraintes imposées dans le cadre du projet et les moyens mis en oeuvre pour le réaliser influencent aussi la façon de le conduire.

Il faut retenir en particulier les paramètres suivants :

* L'étude globale des principaux postes de dépenses (importance relative des salaires d'étude, de la collecte des données pour la construction initiale des fichiers permanents, du budget d'exploitation...) imposera de suivre prioritairement telle ou telle consommation de moyens pour en assurer un contrôle efficace.
* Par ailleurs, au stade de l'analyse organique, il n'est pas possible de spécifier de manière unique une application indépendamment de ses caractéristiques technologiques fondamentales :

= une application temps réel (cf. ODASS [42]) ou une application en traitement différé par lots exigent des standards différents pour expliciter les conditions dynamiques de déclenchement des traitements.

= les structures physiques de données, suivant qu'elles se présentent soit sous forme de fichiers séquentiels ou séquentiels indexés, soit sous forme d'une base de données, ne seront pas définies par les mêmes schémas. Les méthodes d'analyse actuellement proposées sont critiquées parce qu'elles ne permettent pas, dans leur modèle de représentation, la description de bases de données.

= enfin, les langages de programmation utilisés ont une influence sur les normes de rédaction de la logique du traitement : les règles de nomenclatures pour identifier les zones de mémoire, les sous-programmes, les fichiers, sont établies dans le cadre de normes de programmation qui tiennent compte des avantages et inconvénients de chaque langage.

D - La structure de l'équipe de réalisation d'un projet influence aussi directement la méthode de conception d'un projet. Suivant les fonctions distinguées dans le projet et le choix des hommes pour les assurer, la structure de l'équipe varie beaucoup. Par exemple, selon que le chef de projet est un spécialiste de gestion ou un informaticien, selon qu'il existe, ou non, des correspondants informatiques, selon que les membres de l'équipe sont géographiquement dispersés ou au contraire regroupés, le partage des responsabilités dans la rédaction, le contrôle et la diffusion des différentes parties de la documentation ne seront pas toujours assurés de la même manière.

E - Pluralité d'expressions de la logique de traitement des données d'une application

Enfin, il existe de nombreuses formes de spécifications des traitements. Rappelons par exemple :

. les tables de décisions : projet CIVA [13]
. les expressions booléennes : CORIG [11]
. les schémas de programmes [10] [36]
. enfin, les simples ordinogrammes si largement répandus.

Mais, en plus de ces diverses modalités de formalisation des traitements, il faut distinguer, parmi les méthodes d'analyse, celles qui proposent une méthode d'approche du problème par les traitements et celles qui recommandent une approche par les données (voir CXP [11] pp.38 à 41 ou ROCHETTE [35]).

- Les méthodes d'approche par les traitements (CORIG, PROTEE, ARIANE par exemple) ont la séquence de spécification générale suivante :

1. Définition des traitements à assurer
2. Conditions d'enchaînement de ces traitements
3. Données et fichiers nécessaires pour les exécuter et assurer les interfaces entre ces traitements.

- Les méthodes d'approche par les données (MINOS, DELOBEL [12] par exemple) consistent à assurer la séquence de spécification suivante :

1. Définition des données de base saisies et des données de synthèse à produire. Etude de leurs relations logiques.
2. Définition des traitements qui matérialiseront ces propriétés logiques et des structures de données compatibles avec elles.

Evidemment, cette distinction entre traitement et information est partiellement illusoire comme l'a montré ABRIAL [2] avec la notion "d'action spontanée".

1.3. LES OBJECTIFS DE MACSI

Après ces différents rappels, il est possible de situer le projet MACSI en indiquant en quoi ses objectifs diffèrent de ceux qui sont assignés à la plupart des méthodes d'analyse.

Certes le projet MACSI propose comme toute méthode :

- un modèle de représentation d'un système d'information
- des programmes d'aide à l'analyse pour gérer la documentation et contrôler la cohérence des spécifications selon certains critères.

- des possibilités d'enseignement de l'analyse cohérentes avec la méthode.

Cependant, les objectifs essentiels de MACSI qui ont orienté sa conception ont été fixés après un examen des insuffisances constatées dans les méthodes, aujourd'hui disponibles, et compte tenu des difficultés d'enseigner l'analyse à des étudiants en informatique.

Ce sont ces objectifs particuliers du projet MACSI que nous voulons mettre en évidence ici.

1.3.1. Permettre une analyse fonctionnelle d'un projet complète et adéquate

Si presque toutes les méthodes d'analyse disponibles permettent de spécifier complètement l'analyse organique d'un projet et si elles offrent souvent des facilités de programmation intéressantes, par contre elles ne donnent pas au niveau de l'analyse fonctionnelle un modèle de description du projet suffisamment précis pour permettre de décrire complètement une application.

Cette insuffisance est grave car, si un projet est fonctionnellement mal conçu, il conduira à un échec même s'il est réalisé techniquement avec efficacité.

Cette insuffisance des méthodes actuelles s'explique sans doute dans la mesure où elles ont été conçues par des informaticiens pour des informaticiens. Ceux-ci, quelquefois par déformation professionnelle, se préoccupent peu d'étudier à fond un projet avec ses futurs utilisateurs pour examiner quelles fonctions méritent ou non d'être automatisées et quelles seront les difficultés d'organisation administrative pour insérer l'application informatique dans les services. Ils préfèrent étudier les choix techniques de réalisation et des méthodes rapides d'implémentation, quitte à ce que leur production ne corresponde pas complètement aux véritables besoins à satisfaire.

Dans MACSI, nous essayerons de pallier ces difficultés en améliorant l'analyse fonctionnelle d'un projet de deux manières différentes :

1.3.1.1. Permettre aux futurs utilisateurs d'un projet de participer effectivement à la conception d'une partie de ses spécifications fonctionnelles

Pour cela nous proposerons des concepts dans le modèle de description fonctionnelle d'un projet informatique et des règles de description qui seront utilisables par des personnes n'ayant pas une formation informatique approfondie.

Ce modèle et ces règles de description seront définis au chapitre 2.

1.3.1.2. Proposer des outils permettant de construire un prototype d'une application informatique lorsque celle-ci est fonctionnellement difficile à définir.

En effet, dans certains cas, il ne suffit pas de discuter avec les utilisateurs potentiels d'une application pour savoir clairement ce qu'ils souhaitent. Seule une réalisation technique provisoire, utilisée expérimentalement par les utilisateurs, leur permettra de préciser et de hiérarchiser leurs desiderata.

Pouvoir construire le prototype d'une application informatique c'est savoir réaliser dans des délais et à un coût supportable, une autre application informatique qui soit fonctionnellement équivalente à la première : nous proposons dans le projet MACSI des outils permettant de construire un prototype quand le besoin s'en fait sentir. Ces modalités de construction d'un prototype feront l'objet du chapitre 3.

1.3.2. Une méthode d'analyse adaptable aux contraintes particulières de conception propre à chaque projet informatique

Nous avons rappelé au § 1.2.3. la diversité des situations rencontrées dans la conception d'un projet, diversité qui explique le grand nombre de méthodes actuellement proposées aux chefs de projets et analystes.

Or, toutes ces méthodes présentent une grande rigidité. Elles ne sont absolument pas flexibles pour permettre une adaptation de leurs possibilités aux problèmes particuliers que pose chaque projet informatique. Elles sont totalement figées au niveau des formulaires standards de description comme au niveau des programmes d'aide à l'analyse.

A contrario, nous avons essayé dans le projet MACSI de définir une méthode d'analyse, extrêmement flexible dans la mesure où le modèle de représentation d'un système d'information, les règles de description et les programmes d'aide à l'analyse sont facilement modifiables. Notre préoccupation était, à la limite, de dégager une méthode de construction de méthodes d'analyse.

Pour résumer ces deux objectifs fondamentaux de MACSI, nous avons décrit dans la figure 3 quelles seraient, s'ils étaient atteints, les étapes de l'analyse fonctionnelle selon la méthodologie de MACSI.

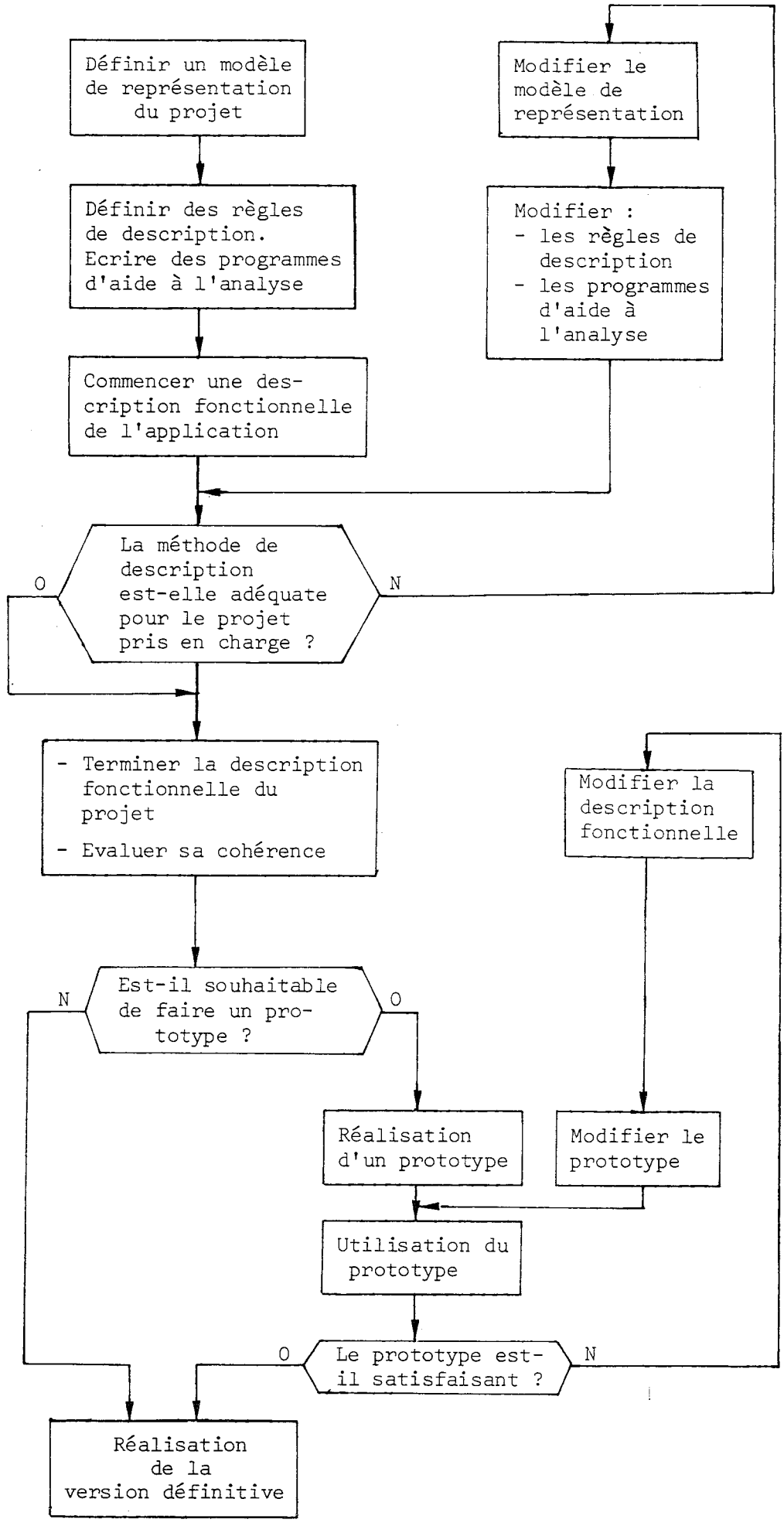


FIGURE 3 - Les étapes de l'analyse fonctionnelle dans MACSI

1.4. HISTORIQUE DU PROJET MACSI - CONSEQUENCES SUR LE PLAN DE CE TRAVAIL

Le projet MACSI est un travail de recherche d'équipe. Il a subi différentes étapes de progression qu'il n'est pas inutile de rappeler dans la mesure où elles expliquent la présentation de ce travail.

- Initialement, le projet MODSIN [30] financé par le DGRST avait pour tâche de définir une méthode d'analyse assistée par ordinateur. Les résultats limités obtenus nous ont permis de prendre conscience des qualités de flexibilité (cf. supra § 1.3.2.) indispensable pour une méthode d'analyse.

- Le projet MACSI a donc été une deuxième tentative accordant la priorité aux objectifs que nous avons définis ci-dessus. Pour cela, nous avons essayé d'adapter certains concepts méthodologiques récents (modularité, analyse descendante) qui sont apparus dans les domaines de la construction des systèmes d'exploitation [22] ou de la programmation structurée [36]. MACSI a aussi largement bénéficié des possibilités nouvelles d'implémentation apportées par le projet SOCRATE. Cet effort de recherche s'est développé en fait selon trois axes parallèles qui correspondent aux trois chapitres suivants.

a) Le projet MACSI1 a été suscité et a largement bénéficié des options du projet MEDOC [20] développé pour le compte du Ministère de la Coopération en vue de définir une méthode automatique de documentation permettant de spécifier les caractéristiques fonctionnelles d'une base de données financières. Dans le cadre de ce projet, les concepts essentiels de MACSI1 ont pu être testés car il a été possible d'évaluer leur utilisation réelle par des analystes, les difficultés pour former ces analystes, et la flexibilité du modèle.

Nous ne présentons dans le chapitre 2 que les principes fondamentaux de MACSI1. A. de CHELMINSKI [7] en a développé une présentation beaucoup plus complète.

b) Le projet MACSI2 a été étudié après différentes expériences de réalisation de maquettes d'applications, rapidement implémentées en Socrate. Ces maquettes devaient faciliter l'étude d'opportunité et l'analyse fonctionnelle de ces applications. Elles ont permis de valider les conditions de faisabilité d'un prototype. Le chapitre 3 présente les techniques d'analyse et de programmation, suggérées par ces expériences, permettant de faciliter la construction d'un prototype.

c) Après la mise au point de plusieurs versions du logiciel de base de données SOCRATE, J.R. ABRIAL a pu présenter un modèle de représentation de la sémantique des données [1]. J.C. FAVRE partait de ce modèle [18] pour implémenter SOCRATEP. Cet effort de formalisation dans le cadre du projet SOCRATE nous a invité à tenter une démarche comparable dans le cadre du projet MACSI. Ce sont les résultats obtenus par cet essai limité d'une synthèse des notions essentielles présentées dans MACSI1 et MACSI2 que nous présentons avec MACSI3 dans le chapitre 4.

Ces trois directions de recherche ne nous apparaissent pas terminées aujourd'hui. D'autres travaux sont en cours qui témoignent d'une évolution possible du projet MACSI. Cette vitalité du projet est aussi un inconvénient dans la mesure où il nous a été très difficile de le présenter sous une forme achevée qui aurait permis de supprimer certaines des redondances ou répétitions que nous n'avons pas pu éviter dans les trois chapitres suivants : que le lecteur veuille bien ne pas nous en tenir rigueur.

0

0 0

CHAPITRE 2

MACSI1

UNE METHODE, ASSISTEE PAR ORDINATEUR, D'ANALYSE FONCTIONNELLE D'UN PROJET

Pour présenter cette méthode d'analyse, nous adopterons le schéma général proposé avec la figure 2 du chapitre 1 en évoquant successivement :

- Le modèle de représentation d'une application informatique de gestion
- Les règles de description
- Le rôle des programmes d'aide à l'analyse
- La flexibilité de la méthode, qui est un objectif prioritaire.

Nous examinerons, pour terminer, les conditions d'extension ou de modification de la méthode et des programmes associés.

2.1. LE MODELE DE DESCRIPTION FONCTIONNELLE D'UN PROJET INFORMATIQUE : APPROCHE QUALITATIVE

Pour décrire la structure du système d'information 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.

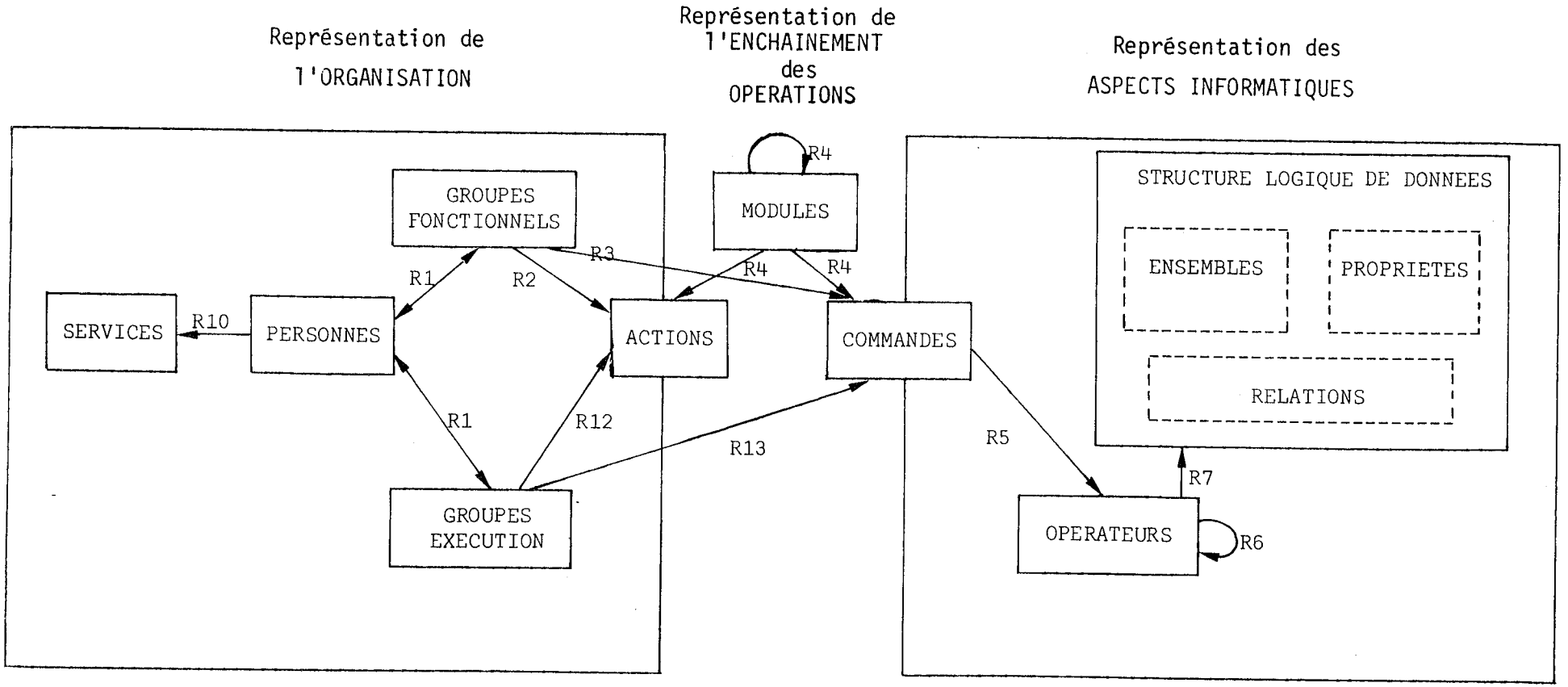


FIGURE 4 - Les divers constituants d'une application informatique

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 4, 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.

1) L'ORGANISATION ENVIRONNANTE

Nous appellerons ACTIONS toutes les opérations qui ont un lien plus ou moins étroit avec le fonctionnement du système d'information. 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 (R10) 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'information. 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 COMMANDES les opérations élémentaires de traitement automatique de l'information qui sont demandées puis utilisées par les personnes de l'organisation.

L'ambiguïté du terme "commande" pour désigner ces opérations, est volontaire parce qu'au niveau de la conception, elles sont perçues comme des "commandes" qui devront être fabriquées par les informaticiens et que plus tard, dans l'utilisation de l'application informatique, elles en seront les "leviers de commande".

Il semble souhaitable de différencier deux types de groupes de personnes dans l'organisation du système d'information.

- 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 COMMANDES (R3) de traitement automatisé de l'information.

- Les GROUPES D'EXECUTION auront la responsabilité, au temps où le projet sera réalisé et disponible, soit de lancer l'exécution d'un programme correspondant à l'implémentation d'une commande (R13), soit d'accomplir une Action (R12).

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 achats 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 commandes 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.

2) L'APPLICATION INFORMATIQUE

Par ses futurs utilisateurs, elle est perçue comme un ensemble de COMMANDES, 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 commande 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 commandes, à préciser ensuite complètement la logique qui sera automatisée.

C'est par des OPERATEURS que sera définie, dans un formalisme sans ambiguïté, la logique des traitements demandés au niveau des commandes. Ils sont spécifiés par des informaticiens. A chaque commande correspondra (R5) un opérateur. Certains opérateurs, d'une complexité suffisante, devront être décomposés eux-mêmes (R6) en opérateurs plus élémentaires.

La STRUCTURE LOGIQUE DES DONNEES définit les relations entre les informations sur lesquelles travaillent les OPERATEURS. 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 (R7) entre les opérateurs et la structure logique des données est définie complètement par la syntaxe qui sera adoptée pour formaliser la logique des opérateurs qui exploitent ces données.

3) ENCHAINEMENT DYNAMIQUE DES OPERATIONS

Commandes et Actions ne s'exécutent pas dynamiquement dans n'importe quel ordre. Les MODULES sont des assemblages (R4) d'actions et de commandes, 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, en obligeant le découpage d'un module en éléments plus simples, 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.

2.2. CHOIX FONDAMENTAUX D'IMPLEMENTATION DE MACSI1

A ce stade, pourtant très sommaire, de définition du modèle, une constatation doit être faite : le nombre de relations entre les constituants du modèle de système d'informations est très important.

Ce fait impose une contrainte au niveau de l'implémentation du fichier documentaire et des programmes de MACSI1.

Si MACSI1 doit être une méthode évolutive, il faut qu'au niveau de la réalisation des programmes d'aide à l'analyse, ceux-ci puissent être facilement modifiés. Pour cela, il faut réduire au maximum la tâche de programmation correspondant à la gestion des pointeurs qui correspondent aux différentes relations du modèle.

Alors que nous avons à peine esquissé l'analyse fonctionnelle de MACSI1, l'évidence et l'importance de cette contrainte sont telles que nous pouvons, dès maintenant, signaler un choix organique fondamental de MACSI1.

La gestion des données et la programmation dans MACSI1 doivent être réalisées avec un logiciel de bases de données évolué : parce que le système SOCRATE était disponible à Grenoble, ce logiciel fut naturellement retenu pour implémenter MACSI1.

Le choix de Socrate comme moyen de réalisation présente par rapport à l'objectif de flexibilité de MACSI1 d'autres avantages induits :

* La structure physique de la base est très évocatrice de la sémantique des relations entre éléments du modèle,

* Les possibilités de modification de la structure de la base (définition dynamique de nouvelles entités de plus haut niveau (cf. réf. [37] p. 29)), sans vidage préalable des données stockées dans cette base, autorisent certaines formes d'extension de l'application,

* Par la clarté et la concision du langage de requête, les programmes écrits avec Socrate explicitent d'eux-mêmes leur logique sans qu'il soit nécessaire de leur adjoindre, pour les documenter, des commentaires nombreux. Ils sont ainsi facilement modifiables.

Remarques : Il existe plusieurs versions assez différentes du système Socrate : celle développée par J.R. ABRIAL et son équipe (réf. [2] [37]) disponible sur IBM 360 sous CP/CMS, celle proposée par CII sur IRIS 45 ou IRIS 50 sous SIRIS 2, celle, enfin, disponible sur IRIS 80 sous SIRIS 7.

Ces trois versions étaient disponibles à Grenoble pour le projet MACSI. MACSI1, compte tenu des contraintes du projet MEDOC dont il est issu, a surtout été programmé sur IRIS 45. Les modèles MACSI2 et MACSI3, présentés dans les prochains chapitres, ont été développés avec la version 4 de SOCRATE sur 360 [37].

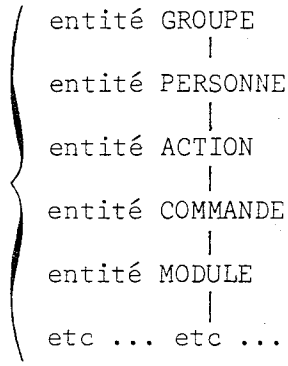
Néanmoins, par souci de cohérence, l'ensemble des structures de données et des programmes présentés dans ce travail seront écrits avec la syntaxe du système Socrate développé sur 360.

2.2.1. Méthode de présentation de la base Socrate utilisée dans MACSI1

Pour poursuivre la description du modèle en définissant les propriétés intéressantes que nous souhaitons retenir pour chaque type de constituant, nous allons alors décrire, en la commentant, la structure elle-même de la base Socrate qui constitue le fichier documentaire de MACSI1.

Dans cette structure, nous allons définir des entités au plus haut niveau, chacune correspondant à un type de composant défini précédemment.

Nous avons ainsi une base documentaire de MACSI1 composée d'entités comme :



En définissant toutes ces entités, leurs caractéristiques et les relations entre elles, nous aurons à la fois spécifié complètement le modèle de description d'un système d'informations et indiqué sur quelle structure de base de données documentaire opèreront les programmes d'aide à l'analyse.

2.2.2. Gestion des nomenclatures et des textes

Avant de décrire les différents constituants d'un système d'informations, rappelons ici un critère de bonne gestion d'un projet que nous avons évoqué précédemment.

- Il est fondamental, dans un projet, que chaque constituant, quel que soit son type, ait un code discriminant vis-à-vis de tous les autres, de façon à pouvoir établir une nomenclature des éléments du projet.

Dans cette perspective, toutes les réalisations des entités GROUPE, PERSONNE, ACTION, COMMANDE, MODULE, OPERATEUR, etc ... devront avoir chacune un code discriminant à la fois :

. vis-à-vis des autres réalisations de leur entité propre,
. vis-à-vis de toutes les réalisations des autres entités.

Pour s'assurer que, dans ce second cas, tout code est bien discriminant, il est créé une entité dictionnaire (DICO) où seront dupliqués tous les codes (CODE) de toutes les entités de plus haut niveau.

La structure de cette entité est ainsi définie :

```
entité 5000   DICO
       début  CODE mot discriminant
         |    entité 20 COMMENTAIRE
         |            début
         |              |   SPECIF référence un EXPLIC
         |            fin
       fin
```

- D'autre part, comme une partie des spécifications se fera dans MACSI1 sous forme de textes (par exemple des libellés, des modes d'emploi, des définitions, etc ...), il est pratique de regrouper toutes les caractéristiques de type texte dans une zone spéciale (entité EXPLIC) sur laquelle pourra opérer spécialement un éditeur de texte. La structure de cette entité est la suivante :

```
entité 1000  EXPLIC
       début
         |   CDEX de 1 à 5000
         |   CONTENU texte 10
         |   entité MOT-CLE
         |        début
         |          |   QUOI référence un DICO
         |        fin
       fin
```

Chaque texte aura un code (CDEX) numérique pour l'identifier. Ce code ne sera pas donné par l'auteur du texte, mais il sera généré de façon interne, au moment de la création d'une réalisation de l'entité EXPLIC, par rapport à une mémoire de travail dans laquelle est rangé le dernier numéro d'explication enregistré.

Toute explication peut être explicative d'un ou éventuellement de plusieurs éléments (par exemple, le texte définissant la fonction d'une personne dans un groupe fonctionnel précise à la fois la PERSONNE et le GROUPE concerné). Aussi, pour faciliter au maximum la recherche documentaire sur les textes, la relation entre tout élément du projet répertorié dans DICO et les textes qui lui sont associés dans EXPLIC est-elle enregistrée dans les deux sens : tous les textes spécifiant un élément sont repérés par (COMMENTAIRE, SPECIF) et réciproquement tout élément indexant un texte

qui s'y rapporte est repéré par (MOT-CLE, QUOI).

2.3. ETUDE DETAILLEE DU MODELE

Compte tenu des conventions de description précédentes, nous allons pouvoir entreprendre un descriptif complet de chaque constituant de base et de l'entité de plus haut niveau qui lui correspond dans la base documentaire.

2.3.1. Les GROUPES et PERSONNES

```
entité 100 GROUPE
       début  CDGR mot discriminant
              TY de 1 à 3
              LIBGR texte 1
              FONCTION référence un EXPLIC
              MEMBRE inverse tout PERSONNE
              entité RESP
                        début ELEMENT référence un DICO
                              PARQUI référence un PERSONNE
                        fin
              COMMEX inverse tout COMMANDE
              ACTEX  inverse tout ACTION
       fin

entité 300 PERSONNE
       début  CDPE mot discriminant
              NOM mot
              PRENOM mot
              entité 5 ROLE
                        début APPARTENANCE référence un GROUPE
                              FONC référence un EXPLIC
                        fin
              ADRESSE texte 10
              TELEPH mot
       fin
```

Rappelons que nous voulons distinguer, dans une application informatique, trois types de GROUPES de personnes :

1 - Les GROUPES de SPECIFICATION qui sont responsables de la conception de l'application (TY = 1).

2 - Les GROUPES D'EXECUTION qui seront chargés d'exécuter, quand l'application fonctionnera, les actions et les commandes qui la composent (TY = 2).

3 - Les UNITES ADMINISTRATIVES qui sont concernées par l'application (TY=3) Chaque personne, identifiée comme membre d'un groupe de spécification et (ou) d'un ou plusieurs groupes d'exécution est rattachée administrativement à une et une seule unité administrative.

I) Quel que soit le type du groupe, il est identifié par un code (CDGR), son libellé (LIBGR), et une définition précise de sa fonction (FONCTION).

Les personnes qui sont rattachées au groupe sont ses (MEMBRES). Réciproquement, une personne peut être rattachée à plusieurs groupes (ROLE) ; son appartenance à chaque groupe est précisée par sa fonction (FONC) dans le groupe. Les différentes relations d'une personne à un groupe doivent respecter les critères suivants :

a) Toute personne doit avoir une appartenance et une seule à une unité administrative qui définit son rattachement hiérarchique dans l'organisation.

b) Quand une personne participe aux spécifications du projet, c'est-à-dire utilise MACSI1, elle peut être rattachée à un ou plusieurs groupes de spécifications.

Remarque méthodologique : Il est souhaitable de distinguer non pas un, mais plusieurs groupes fonctionnels dans la mesure où les responsabilités de chaque personne dans l'équipe de conception d'un projet informatique ne sont pas identiques. A titre d'exemple - car il n'existe pas une seule structure efficace de gestion d'un projet - il est possible de distinguer comme groupes fonctionnels :

- Le chef de projet responsable du projet en termes de délais et de coût.

- Le groupe chargé des spécifications fonctionnelles.

- Le groupe chargé des spécifications informatiques.

- Le coordinateur technique de la réalisation informatique.

- Le groupe des programmeurs chargés de la réalisation.

Chacune de ces fonctions distinguée dans l'équipe doit faire l'objet de la déclaration d'un groupe de spécification dès le début du projet. Chaque groupe de spécification aura, compte tenu de son activité dans le cadre du projet, des possibilités différentes d'utilisation de MACSI1 quant aux accès à la base documentaire et à l'utilisation des programmes d'aide à l'analyse.

c) Après la définition fonctionnelle des actions et commandes confiées à chaque groupe d'exécution, il faudra identifier plus tard,pendant l'analyse organique,les personnes constituant ces groupes de façon à pouvoir organiser les séances de formation de ces personnes aux tâches qui leur sont confiées. Ce n'est qu'après cette formation qu'il est possible de procéder au lancement de l'application. Evidemment, une personne peut appartenir à plusieurs groupes d'exécution.

II) Suivant le type de ces groupes, on peut distinguer certaines propriétés qui leur sont spécifiques.

- Pour les groupes fonctionnels, il est important d'enregistrer toutes les spécifications qu'ils ont fournies (relations R2 et R3 de la figure 4). Dans (RESP), nous repérons les éléments qui ont été déclarés par tous les membres d'un groupe de spécification. Ce repérage est implémenté par pointeur sur les codes de ces éléments dans le dictionnaire et nous identifions la personne qui en a donné les spécifications (PARQUI).

Remarquons que l'association entre les spécifications d'un élément et la personne qui les donne est intéressante uniquement si nous supposons - ce qui est le cas - qu'aucune autre personne ne peut les modifier. Nous contrôlerons ainsi systématiquement, dans MACSI1, que la description d'un élément est exécutée toujours par la même personne tout au long de l'analyse.

- Pour les groupes d'exécution, sont repérées les différentes commandes (COMMEX) dont ils ont la responsabilité d'exécution et, de même, les actions (ACTEX) dont ils sont chargés (cf. relations R12 et R13 de la figure 4).

2.3.2. Les COMMANDES et les ACTIONS

```
entité 500 COMMANDE
        début CDCO mot discriminant
          |     LIBCO texte 1
          |     DEMANDEUR1 référence un PERSONNE
          |     DEFCO  référence un EXPLIC
          |     OPERASS référence un OPERATEUR
          |     EXECUTANT1 inverse tout GROUPE
          |     ETAT de 1 à 10
        fin

entité 200 ACTION
        début CDAC mot discriminant
          |     LIBAC texte 1
          |     DEFACT référence un EXPLIC
          |     DEMANDEUR2 référence un PERSONNE
          |     EXECUTANT2 inverse tout GROUPE
          |     TYPE de 1 à 10
        fin
```

- Chaque COMMANDE, c'est-à-dire chaque demande d'un traitement automatique de données, formulée par un groupe fonctionnel, a un code d'identification (CDCO). Le demandeur (DEMANDEUR1) appartenant à un groupe fonctionnel, est la personne qui formule sous forme de texte libre (DEFCO) la définition de sa commande.

Au cours de l'évolution du projet, cette commande peut être dans différents états de réalisation (ETAT).

* ETAT=1 Elle a été enregistrée mais aucun analyste-informaticien ne s'en est occupé pour en définir précisément la logique.
* ETAT=2 La logique a été définie, sous forme d'un opérateur logique (OPERASS) associé à la commande par un analyste.
* ETAT=3 Il n'y a pas accord réciproque entre le demandeur et l'analyste sur le contenu de la commande. Bien que prise en compte, la commande ne peut pas être logiquement spécifiée.
* ETAT=4 La commande est en cours d'implémentation.
* ETAT=5 La commande implémentée est réceptionnée techniquement sur jeu d'essai (par le coordinateur informatique par exemple).
* ETAT=6 La commande implémentée est réceptionnée par le client demandeur qui donne son accord.

De plus, il faut spécifier éventuellement les divers groupes d'exécution (EXECUTANT 1)qui, dans des circonstances différentes, seront responsables de l'exécution de cette commande. Ils ne sont spécifiés que si l'exécution de l'opérateur associé à cette commande est déclanché à l'initiative d'autres personnes que les membres de l'équipe d'exploitation : c'est le cas en particulier dans certains processus interactifs homme-machine et, plus généralement, de tout traitement optionnel qui n'est exécuté qu'à la demande de services clients.

- Chaque ACTION, comme chaque commande, est identifiée par son code (CDAC).

Est considérée comme ACTION toute opération qui n'est pas, dans le cadre du projet, une commande de traitement automatique de l'information, et qui est indispensable pour que le projet fonctionne. L'ensemble des actions doit permettre de décrire toute l'organisation qui constitue l'environnement amont et aval des procédures informatiques, et par rapport auquel celles-ci prennent leur finalité. Autrement dit, encore, les actions décrivent l'organisation pour laquelle est fabriquée l'application informatique. Cependant, cette première approche de la notion d'ACTION est insuffisante pour permettre une description suffisamment précise de l'organisation environnant une application informatique.

Pour affiner la description d'une action, une possibilité consiste à définir une taxonomie qui apporte une classification, admise par tous les analystes, entre les différents types d'actions. Cette taxonomie doit avoir une double qualité :

- être sans ambiguïté, c'est-à-dire qu'une action déterminée doit être classée sans hésitation par n'importe quel analyste dans une rubrique et une seule de cette classification,

- être pertinente pour la conduite du projet en permettant des diagnostics intéressants sur sa conception.

Le type de chaque action par rapport à la classification proposée est enregistrée dans (TYPE). Cette classification dépend essentiellement du type du projet et du modèle choisi de représentation d'une organisation dans laquelle on introduit le produit informatique.

A titre indicatif, nous proposons la classification suivante pour distinguer les différents types d'actions :

TYPE=1 Décision

= 11 Décision concernant le long terme (plus de 5 ans)

= 12 Décision concernant le moyen terme (de 2 à 5 ans)

= 13 Décision concernant le court terme (dans le cadre de l'année correspondant à l'exercice budgétaire en cours)

= 14 Etude préalable à une décision

TYPE=2 Action de production, de fabrication

= 21 Conception d'un produit ou d'un service

= 22 Fabrication (exécution d'un produit ou d'un service)

= 23 Contrôle qualité du produit

TYPE=3 Contrôle d'exécution

= 31 Contrôle financier

= 32 Contrôle de délai de la production

= 33 Contrôle d'activité des individus

TYPE=4 Action de traitement de l'information non automatisée

= 40 Création d'un document

= 41 Mise à jour d'un document

= 42 Duplication d'un document

= 43 Transcodification d'un document

= 44 Transmission d'un document

= 45 Contrôle manuel du contenu d'un document

= 46 Archivage d'un document.

Cette classification se fonde sur un schéma cybernétique de représentation des organisations, qui est jugé pertinent, à quelques variantes près, par de nombreux auteurs [6] [29].

Il identifie des boucles de régulation : [Décision → Exécution de celle-ci → Saisie et traitement manuel de l'information obtenue à partir des opérations d'exécution → (traitement automatique) → Contrôles à partir de l'information obtenue par les commandes → Nouvelles décisions ou nouvelles opérations].

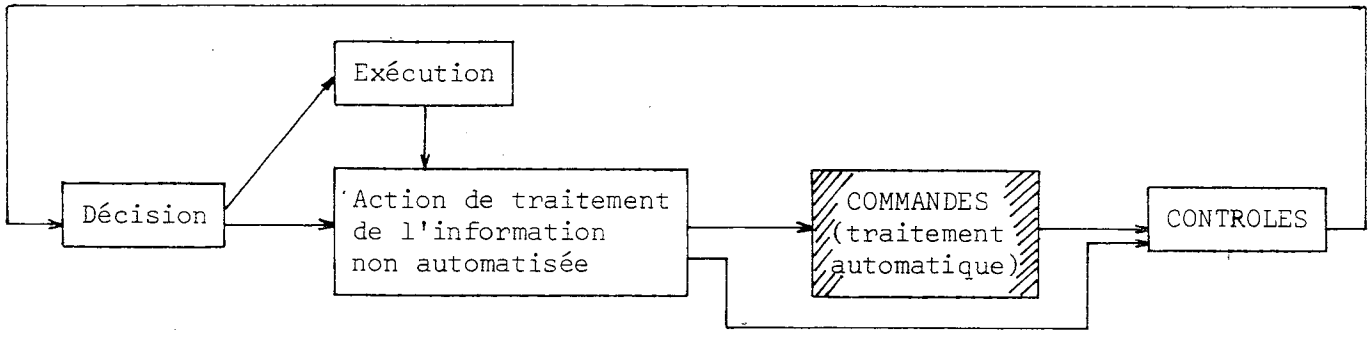


FIGURE 5

En plus du type de l'action, il est important d'indiquer quelles sont la ou les personnes chargées d'exécuter cette action au sein des différents groupes d'exécution (EXECUTANT2) auxquels appartiennent ces personnes (forme réciproque de (ACTEX) dans GROUPE pour matérialiser la relation R12).

La spécification des modalités d'exécution de cette action, y compris le rôle de ces groupes d'exécution, est faite par le demandeur (DEMANDEUR2), sous forme d'un texte (DEFACT).

2.3.3. Les MODULES

Un MODULE est une structure composée de COMMANDES, d'ACTIONS et même de MODULES.

Examinons la structure d'un module dans la base documentaire :

```
entité 50 MODULE
    début CDMO mot discriminant
    |   LIBMO texte 1
    |   DEFMOD référence une EXPLIC
    |     entité 50 ELEMENT
    |           début
    |           |   CDEL mot
    |           |   EXEC référence un GROUPE
    |           |     entité SUITE
    |           |           début CONDITION référence un EXPLIC
    |           |           |   ELEM-SUITE référence un ELEMENT
    |           |           |              de un MODULE
    |           |           fin
    |           fin
    |   DEMANDEUR3 référence un PERSONNE
    fin
```

Chaque Module a un code discriminant (CDMO) pour l'identifier.

Initialement, il doit être décrit globalement et qualitativement sous forme d'un texte libre (DEFMOD). Comme la notion de MODULE dans MACSI1 doit permettre de représenter l'enchaînement dynamique des commandes et des actions, la personne d'un groupe fonctionnel (DEMANDEUR3) qui décrira la structure du module sera aussi chargée ensuite de décrire dans le détail les actions et commandes constituant le module : il faudra vérifier que DEMANDEUR3 d'un MODULE est identique aux DEMANDEUR1 ou DEMANDEUR2 ou DEMANDEUR3 de ses constituants.

Ces différents constituants d'un module s'appellent ses (ELEMENT). Chaque élément est repéré par son code (CDEL) qui est le code ou de l'action (CDAC), de la commande (CDCO) ou du module (CDMO) auquel il correspond. CDEL permet d'implémenter la relation (R4) avec appel par la valeur.

Le groupe d'exécution (EXEC) d'un élément doit être un des (EXECUTANT) de l'action ou de la commande correspondante.

A) Règles de compositions 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 C01 est vérifiée, ou de C si la condition C02 l'est. B et C seront deux réalisations de l'entité SUITE de A ; la première aura dans (CONDITION) le texte définissant C01, la seconde le texte définissant C02.

- Comme C01 et C02 s'expriment par des textes, il faut préciser que C02 n'est pas forcément la négation de C01.

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 ... S_i ... 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 ... 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.

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.

B) 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 CDMO = M1.

L'entrée unique de M1 est M2, ses sorties M9 et C2.

De même, l'entrée du module M4 est A6, les sorties de M4 sont A10, C6 et C1.

CDMO Module	ELEMENT			SUITE	
	EXEC	CDEL	LIBMO (Libellé de définition de cet élément)	ELEM SUITE	CONDITION (éventuellement)
M1 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 C1	Jusqu'à une semaine de l'ouverture : tous les 15 jours. Deux jours avant le congrès
	auto	C1	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 C2	- 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
fin M1	RESPON	C2	Nouveau tirage des listes, compte tenu des participants réels	sortie	Si les comptes sont tous soldés

M4 entrée	SEC	A6	Réception des bulletins d'inscription remplis	A7 A8	
	PERF	A7	Perforation de ces bulletins	C3	
	auto	C3	Mise à jour du fichier des inscrits	C4 C5 C6 C1	En cas d'erreur dans le bulletin En cas de trop payé et de payement insuffisant Facultatif, uniquement à la demande du secrétariat
	auto	C4	Edition d'un message d'anomalie	A7	
	auto	C5	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	C6	Edition de la situation des payements	sortie	
fin M4	SEC	C1	(Déjà vu dans M1)	sortie	

Structure modulaire (suite et fin)

CDGR	LIBELLE (LIBGR)
RESPON	Groupe de personnes responsable de l'organisation générale du congrès
TECH	Groupe chargé de l'organisation matérielle du congrès
IMP	Cellule chargée de l'impression de la documentation publicitaire du congrès
SEC	Secrétariat chargé du suivi des inscriptions
PERF	Groupe de perforation

Liste des groupes d'exécution distingués dans la structure modulaire

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 6.1 ou celle de la figure 6.2 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 6.1) ou bien (figure 6.2) 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 6.2.

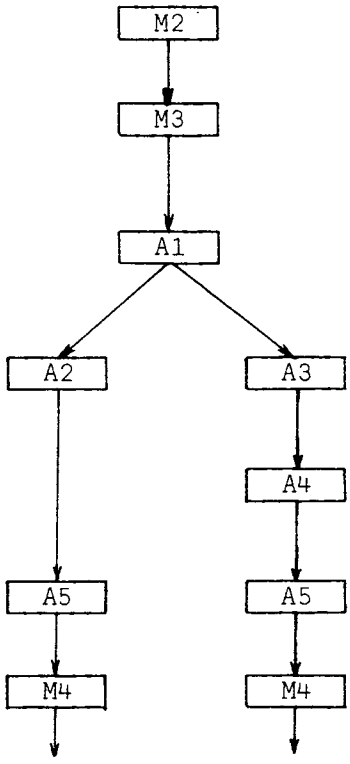


Figure 6.1

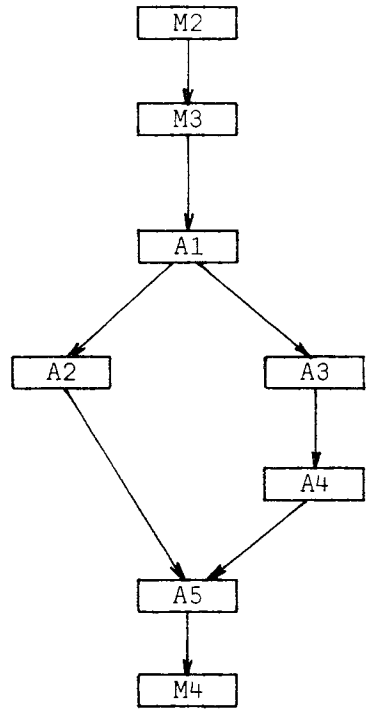


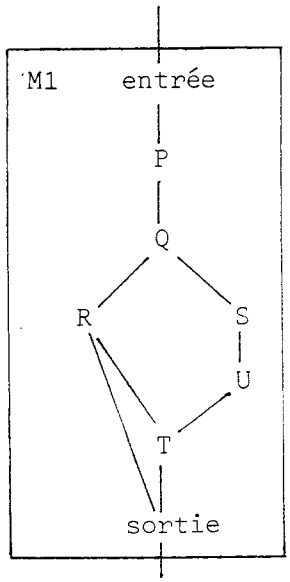
Figure 6.2

Cette difficulté de distinguer les deux cas nous amène à proposer des règles supplémentaires de description de modules.

C) 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.

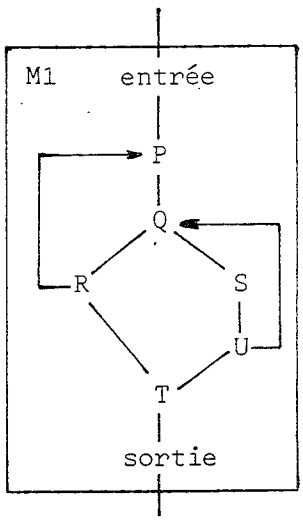
Exemple :



Module	Elément	Elément suite
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 :

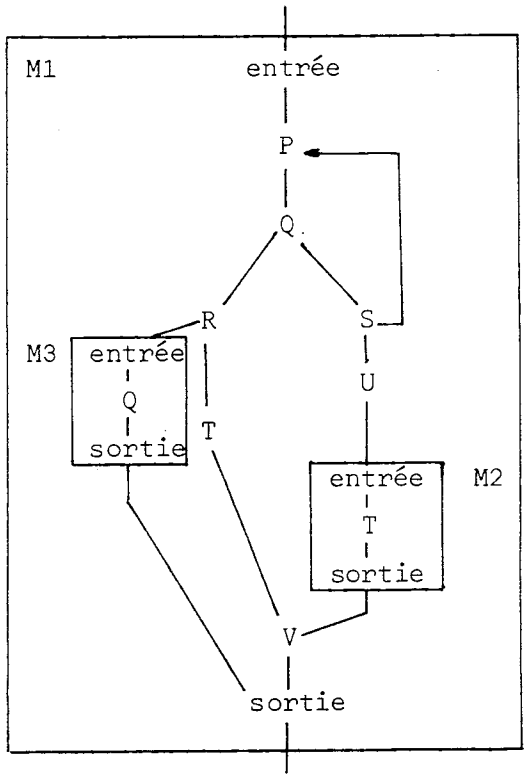


Module	Elément	Elément suite
M1	entrée P	Q
	Q	R, S
	S	U
	U	Q, T
	Q	R
	R	P, T
	T	sortie
fin M1		

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 commande ou un même module.

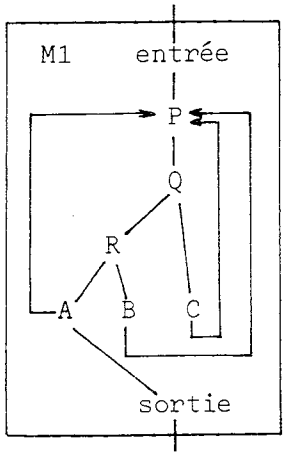
Exemple :



Module	Elément	Elément suite
M1	entrée 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 soient les sorties.

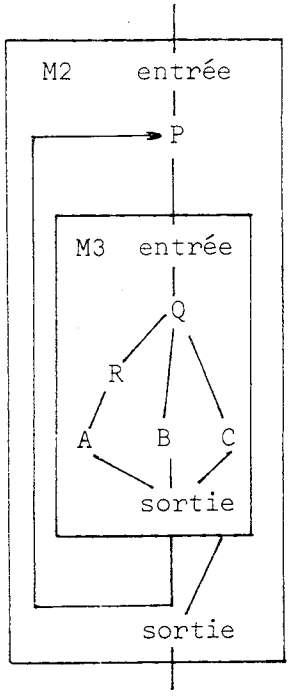
Exemple :



Module	Elément	Elément suite
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	Elément suite
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 commandes, actions et modules entraîne, quelquefois l'obligation de créer, en application des règles 7 et 8, des modules qui, fonctionnellement, sont artificiels.

Réflexions méthodologiques

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 COMMANDES, 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 commandes). 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. [10] [36] ou la méthode ISDOS réf. [40] p. 7 et [39] p. 1121).

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 "commande" 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 [36] : la décomposition d'un programme en module est laissé 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 commande. Ils sont résumés dans le tableau suivant :

QUESTION	Réponse selon la nature de l'élément	
	MODULE	ACTION ou COMMANDE
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 COMMANDES avec des règles d'enchaînement entre celles-ci, mais entre ces opérations, il n'existe aucun interface analogue aux variables d'états retenues dans d'autres modèles de représentation de systèmes.

Ainsi, dans le modèle Forrester [19], entre deux fonctions de décision, doit figurer forcément un niveau.

Ainsi, Horning et Randell (réf. [22])proposent de distinguer, dans la définition d'un processus,un "espace des états" et un "espace des actions" définies comme des transitions d'un état à un autre . On peut identifier ces variables d'état en essayant d'imaginer l'état dans lequel se trouverait le système, si on supposait qu'à un instant donné son fonctionnement s'arrêtait.

Mais dans MACSI1, les interfaces existant entre actions et commandes, telles qu'elles sont définies, peuvent être de nature très diverses :

Nature de l'interface	Entre
Un produit	Deux actions de production (type 2).
Un document	Deux actions de type 4. Une action de type 4 et une commande de saisie de donnée. Une commande d'édition et une action de type 3.
Un fichier Une base de données	Deux commandes.
Un rapport	Une action de type 4 et une action de type 1.
Le plan de fabrication d'une pièce	Une action de type 21 et une action de type 22.

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'enregistrement d'un fichier, etc ...). Aussi, les extensions de MACSI1 pour l'analyse organique, présentées par CHELMINSKI [7],font apparaître les notions de documents et de fichier : le lecteur pourra s'y reporter. Dans un souci de clarté de présentation des fondements de MACSI1,nous n'évoquerons pas ici ces possibilités.

III) Distinction entre actions et commandes

Il peut sembler prématuré de distinguer,au niveau de l'analyse fonctionnelle,des commandes et des actions de traitement manuel de l'information (action de type 4). 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 avec un ordinateur. En conséquence, il est naturel de préciser la nature de certains éléments en tant que "commande" 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.

Aussi, nous mentionnons, dès maintenant, parmi les possibilités offertes par la règle de description MODMO (modification de module, cf. infra § 2.5.3.), celles qui permettent de transformer la structure d'un module selon l'une des deux procédures suivantes :

1) Spécifier que tous les éléments X, correspondant à l'Action X', doivent être associés dorénavant à une commande X'. Ceci se traduira par une mise à jour de la base documentaire réalisée comme suit :

- La Suppression de la réalisation de l'entité ACTION ayant CDAC = X'
- La Création d'une nouvelle réalisation de l'entité COMMANDE ayant CDAC = X'
- Aucune modification des ELEMENTS X dans différents MODULES qui avaient pour CDEL = X'.

2) Réciproquement, il est possible,pour tous les éléments X déclarés correspondre à une commande X', de les associer en fait à une action X'.

IV) <u>Les spécifications des modules pourraient être plus rigoureuses</u> qu'elles ne le sont avec les CONDITION des SUITE 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 (cf. HORNING et RANDELL [22]).

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 MACSI2 présentée au chapitre 3. Mais pour être manipulés correctement, ils nécessitent que les personnes qui s'en servent aient une bonne formation en algorithmique.

Or, rappelons que dans MACSI1 (cf. § 1.3.1.1.), un des objectifs est d'obtenir que les personnes responsables de la spécification générale de l'application (les DEMANDEUR dans ACTION, COMMANDE et MODULE) 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 du concept de MODULE proposé dans MACSI1, au niveau des conditions d'enchaînement des éléments, nous a semblé être 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 commande, mais l'exécution d'une action ou d'une commande.

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 commande 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 commande C1 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.

On comprend ainsi pourquoi,dans les entités COMMANDE et ACTION nous faisons figurer une caractéristique "EXECUTANT inverse tout GROUPE" et non pas la caractéristique "EXECUTANT référence un GROUPE".

2.3.4. STRUCTURE des DONNEES et OPERATEURS : critères de choix d'un modèle de représentation.

Si toute méthode d'analyse se doit de représenter la logique des traitements informatiques à la fois par le choix d'une structure des données et par la définition des algorithmes agissant dessus, il faut constater que dans les méthodes actuelles, il n'existe aucune homogénéité entre les divers formalismes retenus.

Avec l'étude de TEICHROEW [39] sur la représentation des données dans les principales méthodes d'analyse existantes, il faut constater que les recommandations du comité CODASYL n'ont pas de grandes conséquences au niveau des standards adoptés par ces méthodes : elles diffèrent sur le vocabulaire et, même, sur les notions retenues.

Par ailleurs, des modèles théoriques de représentation de la sémantique des données comme ceux proposés par ABRIAL [3] ou DELOBEL [12] ne sont pas identiques.

DELOBEL propose un modèle relationnel pour exprimer une partie de la sémantique d'une collection de données. ABRIAL souligne l'équivalence du langage de requête Socrate avec le calcul des prédicats du premier ordre.

Il existe plusieurs raisons qui expliquent cette diversité a priori surprenante. Nous voulons essayer de les résumer pour situer par comparaison les options que nous avons prises.

2.3.4.1. Pluralité des représentations

a) Interdépendance entre structure logique et structure physique des données.

Quelles que soient les perspectives retenues dans tel ou tel modèle, il y a toujours un mélange de deux types de représentations très différentes :

- d'une part, l'expression de la sémantique du problème pris en compte dans l'application, du seul point de vue fonctionnel, indépendamment des moyens techniques retenus pour implémenter cette sémantique,

- d'autre part, la représentation de la structure physique de données sur des supports et les méthodes d'accès correspondantes.

Plus précisément, au niveau des méthodes de programmation, on étudie particulièrement la gestion de la mémoire. Selon les méthodes d'accès aux données disponibles [10], on peut comparer des structures de listes, de vecteurs ou de tableaux, etc ..., en tant que telles, sans se préoccuper de savoir quelle sémantique elles représentent.

Par contre, dans l'analyse d'une application informatique, il faut d'abord construire un modèle d'un certain univers physique (sémantique), puis implémenter ce modèle par des structures physiques de mémoire et par des programmes : toute la difficulté provient de ce que l'on pense aux deux problèmes en même temps. Le mélange intime entre ces deux approches conduit à toutes sortes de propositions hybrides.

En particulier, au niveau de la définition des données, des hypothèses sont souvent faites, dès le départ de l'étude, sur les types de système de gestion de fichiers disponibles, et, dès lors (cf. dans ISDOS le langage PSL [41]) un formalisme qui se veut descriptif uniquement du problème, indépendamment de tout élément de solution, est, en fait, largement entaché de spécifications organiques telles que la longueur des zones de données ou le choix, pour représenter un graphe, de structures mémoire par matrice ou liste. Des spécifications de ce genre, liées à des choix techniques, sont produites, trop tôt, au niveau de l'analyse fonctionnelle alors qu'elles devraient intervenir au niveau organique seulement. C'est du reste ce que nous avons fait nous-même ici

dans la présentation de MACSI1, en inférant au § 2.2. des choix organiques d'une description fonctionnelle encore très incomplète.

Evidemment, ne sachant pas spécifier de façon automatique, à partir d'un modèle sémantique de données indépendant de tout matériel, une structure physique de données sur une machine déterminée, compte tenu de certains critères de performances, les analystes sont obligés de faire cette spécification de manière heuristique. Cette démarche les conduit souvent à ce que leur choix des structures mémoires soit malheureusement posé trop tôt, avant que la logique du problème ne soit totalement explicitée.

b) Sémantique implémentée soit par les données, soit par programme.

Pour une large part, l'implémentation de la logique d'une application peut être faite soit au niveau des données, soit au niveau des programmes.

Les exemples sont innombrables. Ainsi, dans Socrate, sont disponibles des clauses sémantiques particulières pour la définition d'une base de données : citons, entre autres, les caractéristiques "liste de valeurs" et la limitation des caractéristiques numériques dans une plage de valeur. On peut alors écrire, avec ces instructions au niveau de la structure de la base, toute une partie de la sémantique des contrôles qui seront exécutés dans ce cas par le système SOCRATE lui-même. Mais on peut aussi écrire une structure plus simple et faire exécuter ces contrôles par des programmes spécifiques dont on maîtrisera l'exécution.

La question se pose de savoir si, dans les langages de programmation ou les langages de spécification de la logique d'un problème, il faut enrichir ou, au contraire, appauvrir au minimum les moyens d'expression sémantique incorporés au niveau des clauses descriptives de la structure des données.

D'une discussion avec ABRIAL sur les conclusions qu'il tirait du développement du projet Socrate, j'ai retenu qu'il préférait finalement la seconde solution si du moins, me disait-il, la question avait un sens. Du point de vue du résultat, il remarquait, en effet, qu'il est équivalent de lire la mémoire où sont stockées certaines données, ou d'activer une nouvelle fois le programme qui les avait calculées. Depuis, il a défini la notion d'"action spontanée"[2] dans une base qui précise ce point de vue général.

2.3.4.2. Critères retenus pour définir le formalisme de description des données et des traitements dans MACSI1

Nous souhaitons que l'ensemble des règles qui nous permettront de représenter les propriétés des données et des traitements satisfassent aux critères suivants :

a) La sémantique du problème doit être aussi totalement que possible exprimée avant que les choix d'implémentation soient fixés. Autrement dit, l'analyse fonctionnelle doit être, autant que faire se peut, séparée des choix organiques.

b) Le modèle sémantique des données, inspiré par une approche ensembliste comparable à celle proposée par ABRIAL dans [3], doit être compréhensible par les utilisateurs du projet qui participent à ses spécifications. Il doit donc ne pas proposer une terminologie trop compliquée.

c) Le modèle sémantique retenu pour les opérateurs doit partager les préoccupations des recherches de la programmation structurée actuellement développées [10] [13].

d) Le formalisme des schémas d'opérateurs doit permettre de vérifier une partie de la cohérence de ces schémas de façon automatique. Par ailleurs, il doit permettre d'exprimer,sous une forme unique,des logiques de traitement qui pourront avoir, au niveau de l'implémentation, des modes d'exécution différents :

Création Modification Interrogation	de	Base de données ou fichiers	en temps réel (conversationnel ou non) en traitement par lots

2.3.5. STRUCTURE LOGIQUE des DONNEES

Adoptant une approche ensembliste de représentation des données, nous proposons les définitions suivantes :

A) *DEFINITIONS* : Les données d'une application sont une image d'un univers physique dans lequel nous distinguons des ENSEMBLES de BASE, des PROPRIETES de ces ensembles et des RELATIONS entre ENSEMBLES.

a1) Les ENSEMBLES de BASE sont des ensembles d'éléments concrets (produits, machines, locaux, personnes, clients, etc ...) dont la dénotation est évidente dans l'univers physique que représentent les données. Le repérage de chaque élément d'un ensemble est très souvent fait par un code d'identification discriminant (code produit, numéro de bureau, numéro INSEE, code client, etc ...).

a2) La description de chaque élément d'un ensemble se fait en évaluant les VALEURS de PROPRIETES, spécifiques de l'ensemble, et pouvant être définies pour tout élément de l'ensemble. Le code d'identification d'un ensemble est une de ses propriétés. Chaque propriété P est définie comme une application d'un ENSEMBLE de valeurs V : $E \xrightarrow{P(E)} V$.

Exemple : En continuant à prendre comme illustration des notions présentées, l'application de gestion d'un grand congrès, on peut dans ce cas distinguer comme ensembles :

Ensembles de base	Propriétés de ces ensembles
E_1 Les participants	P_{11}=nom, P_{12}=prénom, P_{13}=adresse
E_2 Les locaux d'hébergement	P_{21}=sigle, P_{22}=adresse, P_{23}=prix journalier
E_3 Les payements	P_{31}=montant, P_{32}=nature du payement (chèque, CCP, liquide)

a3) Il existe, par ailleurs, des RELATIONS BINAIRES entre ensembles : chaque relation est définie comme une partie du produit cartésien de deux ensembles de base E_i et E_j. Une relation binaire $R_K \subset E_i \times E_j$ est donc un ensemble de couples $r \equiv (\ell_{ip}, \ell_{jq})$ où $\ell_{ip} \in E_i$ et $\ell_{jq} \in E_j$.

E_i et E_j seront appelés ARGUMENTS de la relation R_K.

Exemple :

Relation	Argument	Argument	Libellé de la relation
R1	E_1 : Participant	E_2 : local d'hébergement	Réservation effectuée du local pour tel participant
R2	E_1 : Participant	E_3 : payement	Versement effectué par tel participant
R3	E_2 : Local d'hébergement	E_3 : payement	Arrhes versées pour réserver tel logement.

a4) Par extension, chaque relation elle-même peut être considérée en tant qu'ensemble et avoir des PROPRIETES qui lui sont spécifiques.

Exemple :

Relation	Propriété de la relation
R1	P'_{11} : Nombre de jours demandés pour la réservation
R2	P'_{12} : Activités financées par le versement
R3	P'_{13} : Type d'occupation (nuit, demi-pension, pension).

a5) En considérant donc par extension comme des ENSEMBLES (que nous appellerons aussi des ENS-REL : ensembles-relations) soit des ENSEMBLES de BASE, soit des RELATIONS, on peut définir sur ces ensembles de nouvelles relations qui seront des relations de relations.

Ainsi : soit R_i une relation $R_i \subset E_{im} \times E_{in}$

soit R_j une relation $R_j \subset E_{jp} \times E_{jq}$

On peut définir une relation $R_K \subset R_i \times R_j$. C'est l'ensemble des couples de couples $((\ell_\alpha,\ell_\beta), (\ell_\gamma,\ell_\delta))$ où $\ell_\alpha \in E_{im}$, $\ell_\beta \in E_{in}$, $\ell_\gamma \in E_{jp}$, $\ell_\delta \in E_{jq}$ vérifiant la relation R_K et tels que $(\ell_\alpha,\ell_\beta) \in R_i$ et $(\ell_\gamma,\ell_\delta) \in R_j$.

Ainsi, R_K est une relation n-aire (n=4) entre les ensembles de base E_{im}, E_{in}, E_{jp}, E_{jq}.

Exemple :

Relation	Argument	Argument	Libellé
R4	R1	R3	Arrhes versées pour réserver un logement pour tel participant
R5	R2	E_1	Versement effectué par tel participant pour tel autre participant.

Evidemment, des relations de relations peuvent aussi avoir des propriétés.

a6) Finalement, nous ne manipulerons que des **ENSEMBLES** et des PROPRIETES de ces ensembles.

Chaque PROPRIETE est associée de façon unique à un ENSEMBLE.

B) *REFLEXIONS METHODOLOGIQUES*

Le modèle proposé diffère de ceux présentés par ABRIAL et DELOBEL sur les points suivants :

- DELOBEL présente un modèle de collection de données dont la structure est composée de relations n-aires sur des constituants correspondant aux ensembles de base ou aux propriétés.

- ABRIAL propose un modèle composé à partir d' "ensembles" "concrets" ou "abstraits" (correspondant aux ensembles et aux propriétés) et de "relations binaires" entre ces ensembles. A chaque relation binaire sont associées deux "fonctions d'accès".

Les deux modèles ne retiennent pas la notion présentée ici de "propriété" : elle est considérée par eux comme un ensemble en relation avec d'autres.

Cette divergence de point de vue pose le problème méthodologique de savoir s'il existe une différence de nature entre les Ensembles de base, les Propriétés et les Relations.

Pour préciser, constatons qu'il est effectivement possible de représenter, avec le formalisme que nous proposons, le nom d'un participant selon deux modèles distincts :

Premier modèle = M1	Ensemble E_1 "Participant" Propriété P_{11} "Nom", définie comme une caractéristique spécifique de cet ensemble.
Deuxième modèle = M2	Un ensemble E_1 de participants. Un ensemble E_{10} des noms, en tant que tels. Une relation R_{11} ayant pour argument E_1 et E_{10} associant à chaque participant un nom.

Il existe pour nous une différence entre ces deux modèles qui est la suivante :

* Dans le modèle M1, la suppression de l'ensemble E_1 implique celle de toutes ses propriétés, et en particulier de P_{11}, car P_{11} est une propriété spécifique de E_1 et de E_1 seul.

* Dans le modèle M2, par contre, la suppression de E_1 implique la suppression de R_{11} puisque un de ses arguments disparaît, mais l'ensemble E_{10},posé en tant que tel,peut subsister. Ainsi, rien n'empêche, dans les hypothèses du modèle, de mettre en relation E_{10} avec d'autres ensembles

(par exemple tout autre agrégat de personnes).

Cette différence, que nous percevons dans la modélisation d'un certain univers,est donc la suivante :

- Une propriété est une caractéristique associée exclusivement avec un ensemble et un seul. Si, après avoir déclaré son existence, cet ensemble est jugé inadéquat et sa non-existence est affirmée, sont supprimées implicitement toutes les propriétés qui lui étaient associées.

- Par contre, poser l'existence d'une relation R entre deux ensembles E et E' est possible dès que nous pouvons répondre affirmativement aux deux questions suivantes :

. Existe-t-il d'autres relations de E' avec d'autres ensembles que E ?

. La suppression de la relation R entre E et E' conserve-t-elle encore un sens à l'affirmation de l'existence de E' indépendamment de E ?

La réponse négative à l'une de ces deux questions conduit à remplacer l'existence de E' et de R par la déclaration d'une propriété P' de E.

C) *EXTENSION DE LA SEMANTIQUE ASSOCIEE AUX ENSEMBLES, RELATIONS ET PROPRIETES*

Il existe toutes sortes de caractéristiques supplémentaires pour le modèle qui permettent d'enrichir ses possibilités d'expression sémantiques. Examinons celles qui sont retenues dans MACSI1.

c1) POUR TOUS LES ENSEMBLES

Pour tout ensemble, qu'il soit "ensemble de base" ou "ensemble-relation", on peut définir sa taille, c'est-à-dire une borne supérieure du cardinal de cet ensemble durant toute la période pendant laquelle on identifiera les éléments de cet ensemble.

Exemple :

L'ensemble E_1 des participants au congrès n'est pas connu au moment où nous construisons le modèle de gestion de ce congrès. Mais nous pouvons dire que ce congrès, pendant toute la période où nous le gérerons, ne concernera pas plus de 5000 personnes. Nous dirons que E_1 a une taille de 5000.

c2) POUR LES RELATIONS

Pour une relation $\mathcal{R}$ ayant deux arguments E_i et E_j, on peut définir comme le fait J.R. ABRIAL [3] :

* Deux fonctions d'accès : ce sont deux fonctions définies par la relation $\mathcal{R}(E_I,E_j)$ de la manière suivante :

f_1 fonction de E_i dans $\mathcal{P}(E_j)$ telle que

si $\ell_i \in E_i$ et $\forall\ \ell_j \in \mathcal{E}_j \in \mathcal{P}(E_j)$ $\mathcal{E}_j = f(\ell_i) \iff (\ell_i,\ell_j) \in \mathcal{R}$

f_2 fonction de E_j dans $\mathcal{P}(E_i)$ telle que

si $\ell_j \in E_j$ et $\forall\ \ell_i \in \mathcal{E}_i \in \mathcal{P}(E_i)$ $\mathcal{E}_j = f(\ell_i) \iff (\ell_i,\ell_j) \in \mathcal{R}$

Ces deux fonctions d'accès sont intéressantes, car, s'il est quelquefois suffisant de donner le même nom à ces deux fonctions et à la relation, il est aussi parfois souhaitable de distinguer par trois noms différents les deux fonctions et la relation.

Exemple :

E_1 = Personne E_2 = Local $R_1(E_1,E_2)$ = Réservation

f_1 = "réservé-par" : local réservé-par (une personne)

f_2 = "réservant" : personne réservant (un local).

Ces deux fonctions ont, de plus, une propriété qui permet de préciser la relation.

* MIN et MAX d'une fonction d'accès

Pour chaque fonction d'accès, on peut définir le minimum et le maximum du cardinal des sous-ensembles images par f_1 et f_2 des éléments ℓ_i et ℓ_j à un instant donné.

Soient $\mathscr{E}_j = f_1(\ell_i)$ MAX1 = max de $|\mathscr{E}_j|$ $\forall\ \ell_i$

MIN1 = min de $|\mathscr{E}_j|$ $\forall\ \ell_i$

Soient $\mathscr{E}_i = f_2(\ell_j)$ MAX2 = max de $|\mathscr{E}_i|$ $\forall\ \ell_j$

MIN2 = min de $|\mathscr{E}_i|$ $\forall\ \ell_j$

Exemple :

Soit la relation $\mathscr{R}_1$ précédente. A un moment donné quelconque, on peut dire :

MIN1	min de $f_1 = 0$	Combien de chambres au moins une personne peut-elle réserver ?
MAX1	max de $f_1 = 1$	Combien de chambres une personne peut-elle réserver ?
MIN2	min de $f_2 = 0$	Nombre minimum de personnes pouvant réserver une chambre ?
MAX2	max de $f_2 = 2$	Nombre maximum de personnes pouvant réserver une chambre ?

La définition de ces valeurs précise la sémantique du problème de l'hébergement des congressistes sur les points suivants :

Dire que MIN1 = 0 c'est dire que certaines personnes n'ont pas besoin d'être hébergées (cas de l'aller-retour en avion dans la journée).

Dire que MAX1 = 1 c'est dire qu'il n'y a pas de réservation multiple au nom d'une seule personne : elle ne peut faire qu'une seule réservation pour son usage propre.

Dire que MIN2 = 0 c'est constater que certains locaux ne seront pas réservés (offre supérieure à la demande).

Dire que MAX2 = 2 c'est préciser le cas particulier de chambres doubles réservées pour deux personnes (et exclure le cas des couples avec un enfant en bas âge dans la même chambre car,alors,MAX2 = 3).

Remarque méthodologique:

Dans l'exemple précédent, nous avons dit que MAX1 = 1. Comme nous l'avons souligné, nous avons fait l'étude d'invariants sémantiques, comme les MIN et MAX, pour représenter le monde réel observé A UN INSTANT DONNE.

En effet, sur une certaine durée, nous pouvons très bien avoir le cas suivant :

- *A la date t , la personne p réserve le local ℓ_1 : $\mathscr{R}_1(p,\ell_1)$ est vrai*
- *A la date t', la même personne p annule pour convenance personnelle la location ℓ_1 et demande une location ℓ_2 : à ce moment, c'est $\mathscr{R}_1(p,\ell_2)$ qui est vrai et $\mathscr{R}_1(p,\ell_1)$ qui est faux.*

Si nous n'avions pas introduit la clause "à un instant donné", il aurait fallu constater que MAX1 = 2.

c3) POUR LES PROPRIETES

* Le TYPE de VALEUR : nous proposons les types suivants :

TYPE 1 = TEXTE — Les valeurs de ce type sont des textes dont la seule utilité est de transmettre des messages. Il n'existe aucune opération de traitement automatique, sémantiquement signifiante, possible sur ces valeurs.

TYPE 2 = MOT — Ces valeurs sont des chaînes de caractères. L'ensemble des mots n'est doté d'aucune relation d'ordre. Ce sont des chaînes de caractères alphabétiques, numériques ou mixtes pouvant faire l'objet d'opérations de comparaison.

TYPE 3 = LISTE de VALEUR — Ce sont des valeurs de type MOT dont, en plus, on peut préciser une liste de valeurs possibles,invariante au cours du temps.

TYPE 4 = NUMERIQUE [Les valeurs peuvent faire l'objet d'opérations arithmétiques et de comparaison par rapport à la relation d'ordre existant sur les nombres réels.

Evidemment, les valeurs de TYPE 4 sont a fortiori des valeurs de type 2 ou 3.

* COMPOSITION de PROPRIETES : Si P_1 et P_2 sont deux propriétés d'un même ensemble E ayant toutes TYPE = 2 ou 3, on peut définir une propriété P_3 composée de P_1 et P_2 dont chaque valeur, pour un élément donné, est la concaténation de la valeur de P_1 à celle de P_2 pour cet élément (comme exemple : le numéro de Sécurité Sociale).

* Règle d'unicité de la valeur d'une propriété pour un élément d'un ensemble : Nous posons qu'une propriété pour un élément d'un ensemble a toujours une valeur et une seule.Nous verrons l'intérêt de cette contrainte ci-dessous (§ d2). Si certaines propriétés peuvent avoir plusieurs valeurs en apparence (par exemples les prénoms d'une personne), on peut toujours, pour respecter cette contrainte d'unicité, définir plusieurs propriétés (exemple : PRENOM1 PRENOM2 et PRENOM3) et une propriété composée avec les précédentes.

D) QUELQUES POINTS DELICATS DANS LA DESCRIPTION DES DONNEES

Il faut souligner quelques confusions qui peuvent être faites sans une analyse attentive de ce que permet d'exprimer le modèle.

d1) Définition d'un ensemble de base : Il n'est pas toujours évident de définir la fonction caractéristique d'un ensemble de base d'éléments pris dans l'univers réel. Des ambiguïtés peuvent subsister, même en convenant que tout élément de l'ensemble doit avoir une valeur définie pour toutes les propriétés associées à l'ensemble. Une définition précise sous forme de texte, où chaque mot a sa valeur, doit souvent préciser un ensemble en compréhension :

Ainsi, dans l'exemple, comment définir l'ensemble E_1 des participants à un congrès ?

E_1 Est-ce "les personnes qui se sont inscrites à temps ?"

E'_1 Est-ce "les personnes qui se sont inscrites à temps et qui viennent effectivement ?"

E''_1 Est-ce "les personnes qui viennent y compris celles qui s'inscrivent le jour d'ouverture ?"

E'''_1 Est-ce "les personnes qui assistent aux séances techniques ou comprend-on aussi les membres accompagnants ?"

E''''_1 Est-ce "les personnes qui payent pour pouvoir participer, à l'exclusion des personnes invitées ?"

A défaut d'une possibilité de définition en compréhension d'un ensemble, on peut aussi essayer de le définir en extension.

La définition d'un ensemble en extension est alors toujours l'identification de ses éléments par correspondance biunivoque avec les enregistrements d'un fichier : les procédures de saisie de l'information et de mise à jour du fichier sont souvent une définition opératoire des règles de dénotation des éléments de l'ensemble.

Ainsi, si nous définissons E_1^* comme l'ensemble des personnes qui sont identifiées dans le fichier des personnes ayant rempli le bulletin d'inscription, E_1^* est-il un ensemble identique à E_1, E'_1, E''_1, E'''_1 ou E''''_1 ? Seule, une connaissance complète de l'organisation des inscriptions et de la procédure de mise à jour du fichier permet de répondre à la question.

d2) <u>Propriété d'une relation et propriétés de ses arguments</u>

Soient deux ensembles E_1 et E_2

Soit une relation entre E_1 et E_2 : $R(E_1\ E_2)$

Soient pour E_1 des propriétés $P_{11}\ P_{12}\ P_{13}$

Soient pour E_2 des propriétés $P_{21}\ P_{22}\ P_{23}$

Soient pour R des propriétés $P_1\ \ P_2\ \ P_3$

Il faut remarquer qu'il peut être tentant de considérer les propriétés des ensembles comme étant aussi celles de la relation et réciproquement.

Ainsi, la propriété P_1 de R induit une propriété P_{14} de E_1 définie par

$$P_{14}(\ell_j) \equiv P_1((\ell_j\ \ell_k)) \text{ pour } \ell_j \in E_1 \text{ si } R(\ell_j\ \ell_k) \quad \forall\ \ell_k \in E_2$$

Nous voyons alors l'intérêt de la règle posant l'existence pour une propriété d'une valeur et une seule de cette propriété pour chaque élément d'un ensemble.

En effet, si la relation R a, par ailleurs, comme caractéristique MIN2 = 0, on constate que P_{14} n'est pas partout définie sur E_1. Ce n'est donc pas une propriété.

D'autre part, si MAX1 > 1, c'est-à-dire qu'il existe plusieurs $\ell_k \in E_2$ en relation R avec un ℓ_j, P_{14} aura plusieurs valeurs : ce n'est donc pas une propriété de E_1.

Cette contrainte sémantique posée sur la définition d'une propriété permet d'éviter la définition au niveau d'un ensemble d'une propriété qui est, en fait, celle d'une relation dont l'ensemble est un des deux arguments.

E) *STRUCTURE DOCUMENTAIRE POUR DECRIRE LA STRUCTURE LOGIQUE DES DONNEES*

La structure documentaire pour enregistrer la description de la sémantique des données, telle qu'elle a été présentée précédemment, comporte deux entités :

```
entité 200 ENS-REL
        début CDENREL mot discriminant
        |     TY de 1 à 2
        |     DEF1 référence un EXPLIC
        |     PROPR inverse tout PROPRIETE
        |     ARG1 référence un ENS-REL
        |     ACCES1 mot
        |     MIN1 de 1 à 100000
        |     MAX1 de 1 à 100000
        |     ARG2 référence un ENS-REL
        |     ACCES2 mot
        |     MIN2 de 1 à 100000
        |     MAX2 de 1 à 100000
        |     TAILLE de 1 à 100000
        |     PARTICIPE inverse tout ENS-REL
        |     CLE inverse tout PROPRIETE
        fin

entité 500 PROPRIETE
        début CDPROP mot discriminant
        |     DEF2 référence un EXPLIC
        |     TYVAL de 1 à 10
        |     DEQUOI référence un ENS-REL
        |     COMPOSEE (0 N)
        |     COMPOSANT inverse tout PROPRIETE
        fin
```

- (ENS-REL) permet d'enregistrer tous les ensembles de base comme toutes les relations sur les ensembles, considérées elles-mêmes comme ensembles.

- (PROPRIETE) permet de caractériser les propriétés des ensembles et des relations.

Plus en détail, nous notons pour les ENS-REL :

- (CDENREL) est le code d'un ENS-REL.
- Si cet ENS-REL est un "ensemble de base", alors (TY = 1). Si c'est une "relation" alors (TY = 2).
- Une définition, en compréhension, éventuellement en extension, de l'ensemble ou de la relation doit être fournie avec précision dans (DEF1).

- Le cardinal de l'ensemble,ou de la relation,est donné dans (TAILLE). Ses propriétés spécifiques sont identifiées par (PROPR).

- Lorsqu'il s'agit d'une relation (TY = 2), alors sont identifiées les caractéristiques suivantes :

  . ARG1 et ARG2 sont les arguments de la relation
  . Pour chacun de ces arguments (ACCES) est le nom de la fonction d'accès
  . (MIN) et (MAX) sont ceux définis précédemment au § C2.

- Par ailleurs, dans (PARTICIPE) sont identifiées toutes les relations dans lesquelles un ENS-REL intervient comme argument

- Dans (CLE), sont repérées la (ou les) propriété(s) dont les valeurs sont discriminantes pour les éléments de l'ensemble.

Pour chaque propriété, nous enregistrons :

- Son nom (CDPROP)
- Sa définition (DEF2)
- L'ensemble unique auquel elle se rapporte (DEQUOI)
- Son type de valeur (TYVAL) (cf. § C3)
- Si la propriété est composée (COMPOSE = 0), alors sont indiquées les propriétés composantes dans (COMPOSANT).

F) *CONCLUSION*

Ce modèle pour décrire la structure logique des données pourrait être plus complet. On aurait pu par exemple donner d'autres extensions sémantiques comme :

- Distinguer, dans les propriétés numériques (TYPE = 4), celles dont les valeurs sont soit des entiers naturels (TYPE = 41), soit des entiers relatifs (TYPE = 42), soit des rationnels positifs (TYPE = 43), soit enfin des booléens (TYPE = 44).

- La notion de "relation fonctionnelle" proposée par DELOBEL [12].

Les limites du modèle sémantique présenté ont en fait été dictées par le souci de voir, si possible, des personnes sans compétence approfondie en mathématiques et informatique, spécifier elles-mêmes la logique des données de leur problème. Notre expérience retirée d'un enseignement de l'analyse à des auditoires très variés, tout comme notre participation à plusieurs projets informatiques nous ont amené à limiter les caractéris-

tiques du modèle sémantique des données aux propositions précédentes. Ceci n'exclut pas des extensions possibles si elles s'avèrent compréhensibles pour ceux qui les utiliseront.

2.3.6. Les opérateurs

Dans le cadre de la sémantique des données définie précédemment, il vient immédiatement à l'esprit la possibilité de définir certaines opérations sur les ensembles, leurs relations et leurs propriétés.

Il est évident que peuvent être définis de nouveaux ensembles ou de nouvelles propriétés grâce à des opérations sur les ensembles et propriétés existantes, comme par exemple :

- l'union, l'intersection de plusieurs ensembles,
- des relations d'équivalences ainsi définies : soit un ensemble E, une propriété P de E. Soient e $\in$ E, e' $\in$ E. Une relation d'équivalence r est induite par une propriété P de E : e r e' <=> P(e) $\equiv$ P(e'),
- la projection de relations binaires suivant l'un ou l'autre de leurs arguments,
- les opérations arithmétiques sur les valeurs des propriétés numériques,
- etc ... etc ....

Mais plutôt que définir la collection des opérations primitives autorisées et leurs règles de composition, il faut rappeler quelles seront les personnes chargées de spécifier les opérateurs et les résultats attendus de cette spécification en précisant certains critères évoqués au § 2.3.4.2..

La définition des opérateurs est réservée, dans le cadre de MACSI1, aux informaticiens qui participent au projet. Les utilisateurs de ce projet en effet, sans formation informatique particulière, ont des responsabilités de conception dans la partie informatique du projet limitée à la spécification des COMMANDES sous forme de texte libre (DEF COMM) dans lequel seront exprimés les traitements souhaités sur les données. Evidemment, dans ces textes ils peuvent faire référence aux identificateurs des propriétés et des ensembles (CDENREL,ACCES1,ACCES2,

CDPROP). Ils doivent donc collaborer aussi à la spécification des ENS-REL et de leurs PROPRIETES sans se soucier de savoir quelles structures physiques de données seront retenues au niveau organique.

Disposant donc de ces spécifications, les analystes-informaticiens doivent avoir à leur disposition un langage de description des traitements qui permette d'exprimer complètement la logique de ceux-ci.

Le formalisme retenu dans MACSI1, parmi beaucoup d'autres possibles, est le langage de requête Socrate.

Il nous faut expliquer pourquoi nous suggérons dans MACSI1 de retenir comme langage descriptif des opérateurs, le langage de programmation SOCRATE tout comme, dans le domaine des algorithmes numériques, nombreux sont ceux qui retiennent ALGOL pour les présenter.

2.3.6.1. Critères retenus pour choisir le langage de requête SOCRATE comme formalisme d'expression des opérateurs

a1) Exclure toute représentation graphique

Nous avons écarté la représentation sous forme d'organigramme comme on en trouve encore tant dans les dossiers d'analyse traditionnels. Les inconvénients de cette forme d'expression d'un algorithme sont bien connus :

- La lecture en est difficile quand l'algorithme est complexe.
- Il est impossible de rentrer directement dans le fichier documentaire de MACSI ce genre de représentation graphique.
- La mise à jour des organigrammes est très lourde et conduit souvent à une maintenance des dossiers d'analyse rarement exécutée à temps.

a2) Ce formalisme devait permettre de respecter les recommandations faites dans les ouvrages de programmation structurée [10] [36] ; en particulier l'utilisation des règles [si ... alors ... sinon], [tantque ... faire ... refaire] doit être possible . Cette préoccupation nous a été suggérée par les efforts récents d'enseignement de l'algorithmique dans

cette optique de programmation structurée : il n'est pas pédagogique, en effet, de donner aux étudiants l'impression que les règles de construction de programmes qu'ils apprennent dans les cours d'algorithmique n'ont plus d'intérêt quand il s'agit de concevoir une application informatique dans des cours d'analyse.

Il est inutile de démontrer pourquoi la syntaxe de Socrate se prête bien à la programmation structurée. Soulignons, de plus, que l'écriture de programmes en Socrate est assez concise, et pour que ces programmes soient autodocumentés, peu de commentaires sont nécessaires.

a3) Le langage devait être implémenté pour que soit toujours possible une évaluation de la correction syntaxique et sémantique des opérateurs décrits en exigeant qu'ils soient testés sur un jeu d'essai.
Il est, en effet, très difficile de vérifier par simple lecture la cohérence logique d'un jeu de schémas de programmes volumineux : le responsable technique du projet, pour avoir quelques garanties, doit y passer un nombre d'heures impressionnant. Par contre, la sanction d'une impossibilité d'exécution sur ordinateur d'un algorithme mal ou incomplètement spécifié est certainement une forme de contrôle de qualité des plus efficaces.

Socrate a donc été choisi de préférence à d'autres formalismes de programmation structurée qui ne sont pas implémentés et écartent donc cette possibilité de contrôle de la qualité des spécifications produites.

a4) Socrate, enfin, possède un avantage essentiel qui, à notre connaissance, lui est spécifique et s'avère à l'expérience très intéressant : c'est l'instruction d'attachement (cf. réf. [37] pp. 65 à 70).

Rappelons que les instructions [ATT <périphérique logique> <périphérique physique>] et [DET <périphérique logique>] permettent d'attacher les périphériques logiques aux périphériques physiques, ou de les détacher.

Les périphériques logiques sont les différentes formes de communications fonctionnelles entre la machine et ses utilisateurs. Ce sont, entre autres :

- ORDRE = flux des ordres permettant d'activer les opérateurs disponibles
- ENTREE = flux des données d'entrée pour créer ou modifier la base
- SORTIE = flux des données éditées.

Les périphériques physiques sont les organes d'entrée/sortie disponibles sur la machine. Ils s'appellent : CONSole IMPrimante, lecteur [illegible]rateur de CARTes, fichier CMS, fichier SOCRATE, **BANDE**, et même RIEN [illegible]hérique physique virtuel auquel est relié un périphérique logique [illegible]l est débranché !).

Cette distinction entre la notion fonctionnellle de périphérique [illegible]e et la notion organique de périphérique physique apporte, au méthodologique, une possibilité très importante :

Après avoir écrit un opérateur qui explicite la logique d'un problème, il est possible de simuler plusieurs hypothèses organiques de mise en oeuvre en faisant varier ses attachements.

Par exemple, pour un même opérateur, on peut simuler des situations telles que celles du tableau ci-dessous.

ENTREE	SORTIE	ORDRE	TYPE D'UTILISATION
BANDE	IMP	SOCRATE ou CMS	Traitement par lots rencontrés dans plus de 50 % des applications actuelles
CONS	IMP	SOCRATE ou CMS	Saisie on line - Edition temps différé
CONS	CONS	CONS	Usage conversationnel en mode maître
CONS	CONS	SOCRATE	Usage conversationnel en mode esclave

Cette possibilité offerte par Socrate invite les analystes à ne pas s'occuper trop tôt de choix organiques mais à écrire plutôt les opérateurs à un niveau très fonctionnel.

2.3.6.2. Liaison entre opérateurs et structure des données

Après avoir spécifié le modèle sémantique de données, comment écrire des opérateurs en Socrate qui soient exécutables ?

Pour cela, il faut évidemment définir d'abord une structure Socrate de données qui soit compatible avec le modèle sémantique. Or, il est évident qu'à une structure sémantique des données peuvent correspondre de nombreuses structures Socrate qui diffèrent au moins par les méthodes d'accès retenues. Deux procédures sont envisageables :

1) Procédure souhaitable pour construire une structure de données Socrate (figure 7)

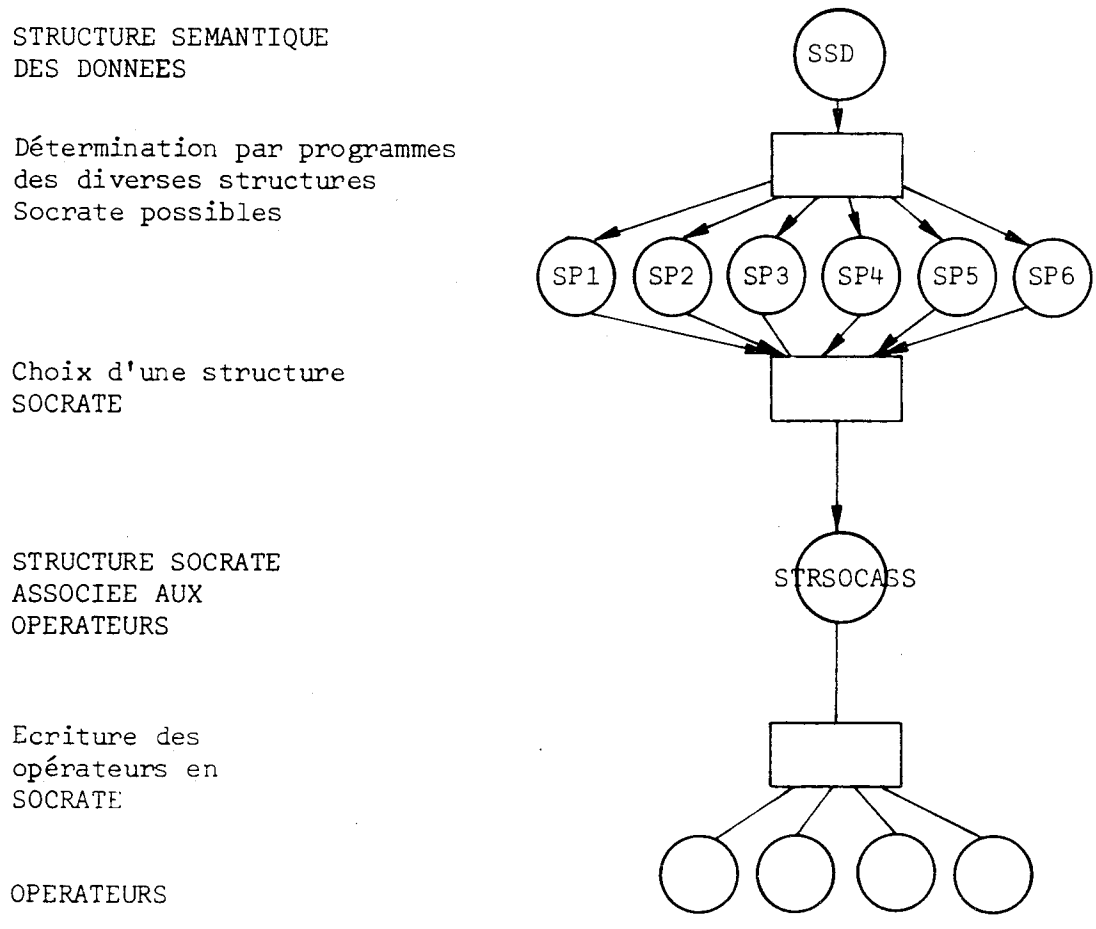


FIGURE 7

On pourrait espérer la construction assistée par ordinateur d'une structure Socrate en construisant un programme de traduction du modèle sémantique qui appliquerait certaines règles automatisables. La grosse difficulté est de nature combinatoire car, pour chaque relation du modèle sémantique, il existe plus de 10 possibilités différentes d'implémenter la relation dans une structure Socrate.

Exemple :

Soient, en effet, deux ensembles E1 et E2 ayant comme clés CL1 et CL2. On suppose que TAILLE de E2 = 1000 et TAILLE de E1 = 5000.

Soit une relation entre eux, REL, dont les deux fonctions d'accès R1 et R2 sont telles que MAX1 = MAX2 = 1 et TAILLE de REL = 500.

Voici quelques-unes des structures Socrate possibles associées à cet exemple.

```
entité 5000 E1
      début CL1 mot discr
        |     R1 référence un E2
      fin

entité 1000 E2
      début CL2 mot discr
        |     R2 référence un E1
      fin
```

Cas 1

```
entité 5000 E1
      début CL1 mot discr
        |     R1 mot
      fin

entité 1000 E2
      début CL2 mot discr
        |     R2 référence un E1
      fin
```

Cas 2

```
entité 5000 E1
      début CL1 mot discr
        |     R1 référence un E2
      fin

entité 1000 E2
      début CL2 mot discr
        |     R2 mot
      fin
```

Cas 3

```
entité 5000 E1
      début CL1 mot discr
        |     R1 de 1 à 1000
      fin

entité 1000 E2
      début CL2 mot discr
        |     R2 référence un E1
      fin
```

Cas 4

```
entité 5000 E1
      début CL1 mot discr
      |     R1 mot
      fin

entité 1000 E2
      début CL2 mot discr
      |     R2 de 1 à 5000
      fin
```

Cas 5

```
entité 5000 E1
      début CL1 mot discr
      |     R1 référence un REL
      fin

entité 1000 E2
      début CL2 mot discr
      |     R2 référence un REL
      fin

entité 500 REL
      début
      |     R11 référence un E1
      |     R12 référence un E2
      fin
```

Cas 6

Il existe,dans le cas où le MAX de la fonction d'accès d'une relation est égal à 1, trois méthodes d'accès disponibles :

1) existence d'un pointeur (référence) (cf. cas 1 R1 et R2, cas 2 R2, cas 3 R1, cas 4 R2)

2) identification de l'élément par la valeur de son code discriminant.

Ainsi, dans le cas 2, R1 a,pour valeur,la valeur de la clé CL2 de la réalisation de E2 repérée par R1 (aussi dans cas 3 R2, cas 5 R1).

3) identification de l'élément par son numéro d'occurrence dans la suite des réalisations de l'entité (voir réf. [37] p. 52)
(cas 4 R1 et cas 5 R2).

Pour l'exemple ci-dessus, on voit qu'il existe en fait 9 cas différents pour la seule réalisation de REL par ses deux fonctions d'accès dans E1 et E2. Il existe d'autres structures possibles comme le cas 6.

Si, de plus, le MAX d'une ou des deux fonctions d'accès d'une relation est supérieure à 1, il est envisageable d'utiliser la méthode d'accès par fichier inverse, ce qui augmente encore le nombre de combinaisons possibles.

Le nombre considérable de structures Socrate possibles pour une structure sémantique des données imposerait donc de définir des règles heuristiques permettant de sélectionner parmi toutes ces structures possibles, celles qui conviendraient le mieux au vu de certains critères de choix : rapidité d'exécution, encombrement mémoire, possibilité de transformer la structure, etc ...

Ce problème est très difficile à résoudre et il n'a pas été pris en compte dans MACSI 1, car il justifiait une très lourde étude à lui seul. Quelques éléments de solution sont cependant proposés au chapitre 4 avec MACSI3.

Il faut donc envisager une autre procédure de construction de la structure SOCRATE.

2) Procédure réalisable pour concevoir la structure Socrate associée (figure 8)

Dès lors, le choix de la base Socrate retenue est un choix, non automatisé,réalisé directement par le coordinateur informatique du projet, à partir du modèle sémantique des données complètement spécifié. Les méthodes d'accès retenues doivent être pratiques pour l'écriture des opérateurs qui seront associés aux commandes exprimées par les utilisateurs.

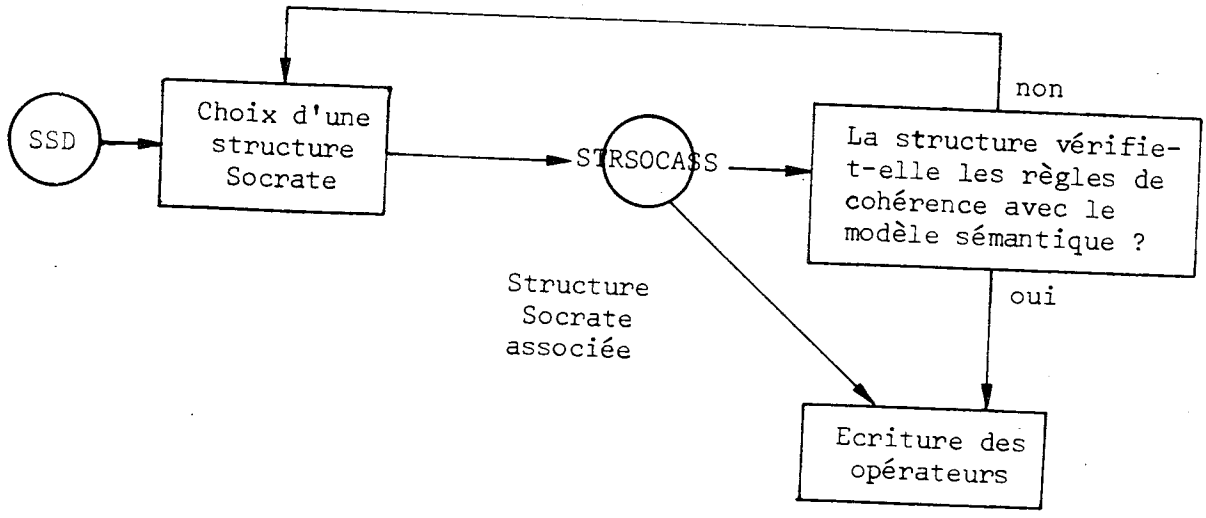


Figure 8

Il faut, pour s'assurer que la structure retenue est compatible avec le modèle sémantique des données, vérifier en particulier le respect des règles suivantes :

- aux noms des ensembles de base correspondent les noms d'entités de plus haut niveau dans la structure Socrate,

- à chaque identificateur de caractéristique d'une entité correspond soit un CDENREL, soit une fonction d'accès ACCES1 ou ACCES2, soit des variables internes de travail. Il faut que ces dernières soient reconnaissables, par exemple, en exigeant que leur symbole commence par un "T-" ou un "W-",

- le type de chaque caractéristique simple doit être cohérent avec celui de la propriété correspondante,

- le nombre de réalisations maximum d'une entité doit être égal soit à la TAILLE d'un ENSREL, soit au MAX1 ou MAX2 d'une relation.

2.3.6.3. Documentation sur les opérateurs

Finalement, les spécifications concernant les opérateurs seront stockées dans le fichier documentaire sous la forme suivante :

```
STRSOCASS texte 400

entité 500 OPERATEUR
      début CDOP mot discriminant
            DEF3 référence une EXPLIC
            OPER-FILS inverse tout OPERATEUR
            OPER-PERE inverse tout OPERATEUR
            SOURCESOCRA référence une EXPLIC
            ESPDONNEE inverse tout PROPRIETE
            COMM référence une COMMANDE
      fin
```

Une fois la structure Socrate associée au modèle sémantique des données (STRSOCASS) fixée par le coordinateur informatique, les opérateurs (OPERATEUR) qui vont être écrits auront un code discriminant (CDOP) et un libellé (DEF3) qui indique en quelques lignes quelle est leur fonction. Le texte Socrate de chaque opérateur sera dans (SOURCESOCRA) lorsqu'il aura été testé en compilation et en exécution sur un jeu d'essai. Par construction,

un opérateur peut être appelé par plusieurs autres opérateurs (OPER-PERE), de même que, réciproquement, un opérateur peut en appeler plusieurs (OPER-FILS). Les caractéristiques de la structure Socrate associée sur lesquelles agit un opérateur, que ce soit en lecture, mise à jour, création ou suppression, sont identifiées dans (ESPDONNEE).

Enfin, à chaque commande demandée par les gestionnaires, correspond un opérateur. Dans celui-ci, cette commande est précisée par COMM.

2.4. RECAPITULATIF DE LA STRUCTURE DU FICHIER DOCUMENTAIRE

Arrivé à ce point, nous avons vu l'ensemble des caractéristiques du modèle MACSI1 qui doivent être spécifiées pour que l'analyse fonctionnelle d'une application informatique soit terminée. Nous pouvons la résumer ici pour faciliter la lecture de la suite de ces pages.

```
entité 1000 DICO
      début CODE mot discriminant
        |     entité 20 COMMENTAIRE
        |            début
        |              | SPECIF référence un EXPLIC
        |            fin
        |
      fin

entité 5000 EXPLIC
      début CDEX de 1 à 5000
        |     CONTENU texte 20
        |     entité MOT-CLE
        |            début QUOI référence un DICO
        |              |
        |            fin
        |
      fin
```

```
entité 100 GROUPE
      début CDGR mot discriminant
        |     TY de 1 à 3
        |     LIBGR texte 1
        |     FONCTION référence un EXPLIC
        |     MEMBRE inverse tout PERSONNE
        |     entité 50 RESP
        |            début ELEMENT référence un DICO
        |              |     PARQUI référence un PERSONNE
        |            fin
        |     COMMEX inverse tout COMMANDE
        |     ACTEX inverse tout ACTION
      fin

entité 300 PERSONNE
      début CDPE mot discriminant
        |     NOM mot
        |     PRENOM mot
        |     entité 5 ROLE
        |            début APPARTENANCE référence un GROUPE
        |              |     FONC référence un EXPLIC
        |            fin
        |      ADRESSE texte 10
        |      TELEPH mot
      fin
```

```
entité 100 MODULE
      début CDMO mot discriminant
        |     LIBMO texte 1
        |     DEFMOD référence un EXPLIC
        |     DEMANDEUR3 référence un PERSONNE
        |            entité 50 ELEMENT
        |                   début CDEL mot
        |                     |     EXEC référence un GROUPE
        |                     |    entité 10 SUITE
        |                     |      début CONDITION référence un EXPLIC
        |                     |        |     ELEM-SUITE référence un ELEMENT
        |                     |        |        de un MODULE
        |                     |      fin
        |                   fin
      fin

entité 500 COMMANDE
      début CDCO mot discriminant
        |     LIBCO texte 1
        |     DEFCO référence un EXPLIC
        |     DEMANDEUR1 référence un PERSONNE
        |     OPERASS référence un OPERATEUR
        |     EXECUTANT1 inverse tout GROUPE
        |     ETAT de 1 à 10
      fin

entité 200 ACTION
      début CDAC mot discriminant
        |     LIBAC texte 1
        |     DEFACT référence un EXPLIC
        |     DEMANDEUR2 référence un PERSONNE
        |     EXECUTANT2 inverse tout GROUPE
        |     TYPE de 1 à 100
      fin
```

```
entité 200 ENS-REL
      début CDENREL mot discriminant
        |   TY de 1 à 2
        |   DEF1 référence un EXPLIC
        |   PROPR inverse tout PROPRIETE
        |   ARG1 référence un ENSREL
        |   ACCES1 mot
        |   MIN1 de 1 à 100000
        |   MAX1 de 1 à 100000
        |   ARG2 référence un ESNREL
        |   ACCES2 mot
        |   MIN2 de 1 à 100000
        |   MAX2 de 1 à 100000
        |   TAILLE de 1 à 100000
        |   PARTICIPE inverse tout ENS-REL
        |   CLE inverse tout  PROPRIETE
      fin

entité 500 PROPRIETE
      début CDPROP mot discriminant
        |    DEF2 référence un EXPLIC
        |    TYVAL de 1 à 10
        |    DEQUOI référence un ENSREL
        |    COMPOSE  (0 N)
        |    COMPOSANT inverse tout PROPRIETE
      fin
STRSOCASS texte 400

entité 500 OPERATEUR
      début CDOP mot discriminant
        |    DEF3 référence un EXPLIC
        |    OPER-FILS inverse tout OPERATEUR
        |    OPER-PERE inverse tout OPERATEUR
        |    SOURCESOCRA    référence un EXPLIC
        |    ESPDONNEE inverse tout PROPRIETE
        |    COMM référence un COMMANDE
      fin
```

CONCLUSION : Il est évident que nous venons, en fait, de présenter depuis le début de ce chapitre, la structure Socrate STRSOCASS associée à MACSI1. Le concept n'étant pas défini, nous ne pouvions pas donner le modèle sémantique des données de MACSI1 auquel elle correspond. Mais le lecteur aura deviné lui-même quelles sont, dans le modèle sémantique de MACSI1,

pour les Ensembles de base { DICO, EXPLIC, GROUPE, PERSONNE, MODULE, COMMANDE, ACTION, ENS-REL, PROPRIETE, OPERATEUR } leurs propriétés et leurs relations ?

Nous avons défini les ensembles de base identifiables dans un système d'information, leurs propriétés et leurs relations servant à spécifier l'analyse fonctionnelle d'un projet informatique, par la présentation de la structure Socrate associée à MACSI1.

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

a) Sous quelles formes et selon quelles règles faut-il rédiger les spécifications d'un projet qui sont chargées dans cette base documentaire ? C'est l'objet du paragraphe 2.5 suivant.

b) Selon quels critères, certains diagnostics automatiques seront-ils possibles avec ce modèle de description d'un projet informatique ? Ils seront examinés dans le paragraphe 2.6.

c) Quelles sont les possibilités d'extension du modèle présenté ? Nous verrons les possibilités de modifications de MACSI1 au paragraphe 2.7.

2.5. REGLES DE DESCRIPTION ASSOCIEES AU MODELE

Pour rédiger les spécifications d'un projet conformément au modèle de description, il est proposé, dans les méthodes d'analyse existantes, un jeu de formulaires standards préimprimés. Chaque formulaire correspond en général à la description d'un type de composant de base du modèle. On trouvera dans [11] certains fac-similés de formulaires.

Si nous appliquions la même démarche, il faudrait dessiner une famille de bordereaux comme par exemple :

bordereaux descriptifs
- de modules
- d'actions
- de commandes
- de groupes
- de personnes
- d'ensembles et de propriétés, etc ...

Chacun de ces bordereaux demanderait un temps considérable de conception graphique et sa structure conditionnerait complètement le programme de mise à jour de la base correspondant. Le bordereau descriptif des modules pourrait par exemple ressembler à celui présenté page 2.20.

Cette solution ne serait pas satisfaisante, compte tenu essentiellement de l'objectif de flexibilité assigné à MACSI1.

En effet, toute modification dans le modèle d'analyse, soit par transformation, ajout ou suppression d'une propriété d'un constituant, soit par modification d'une relation, soit même par création d'un nouveau type de constituant, entraînerait forcément un travail considérable de maintenance puisqu'il faudrait :

- redessiner complètement le (ou les) bordereau(x) concerné(s),
- reécrire complètement le (ou les) programmes(s) de contrôle des spécifications et de mise à jour de la base associé(s) à ce(s) bordereau(x).

Il faut donc proposer un autre principe de description, que celui du remplissage de bordereaux préimprimés ; il doit satisfaire deux conditions :

1) Les règles de description doivent être très facilement modifiables en cas de modification du modèle, ainsi que les programmes correspondants de contrôle et mise à jour de la base documentaire.

2) L'utilisation de ces règles de description doit être facilement compréhensible de leurs utilisateurs pour ne pas alourdir le travail de formation des analystes qui s'en serviront.

2.5.1. Règles de description

Dans MACSI1, les spécifications d'un projet sont écrites en utilisant un formalisme défini par des REGLES de DESCRIPTION (notées par la suite RD).

a) Définition : Une RD (règle de description) est un automate d'état fini exprimant l'ordre dans lequel sont introduites les valeurs des propriétés et relations associées à un type de constituant.

Etant donné les constituants de base du modèle sémantique de MACSI1 auxquels correspondent les entités de plus haut niveau dans la structure Socrate de la base documentaire, les noms des différentes règles de description correspondant à une opération soit de création, soit de mise à jour, soit de modification d'une réalisation d'une de ces entités de plus haut niveau sont résumés dans le tableau ci-dessous.

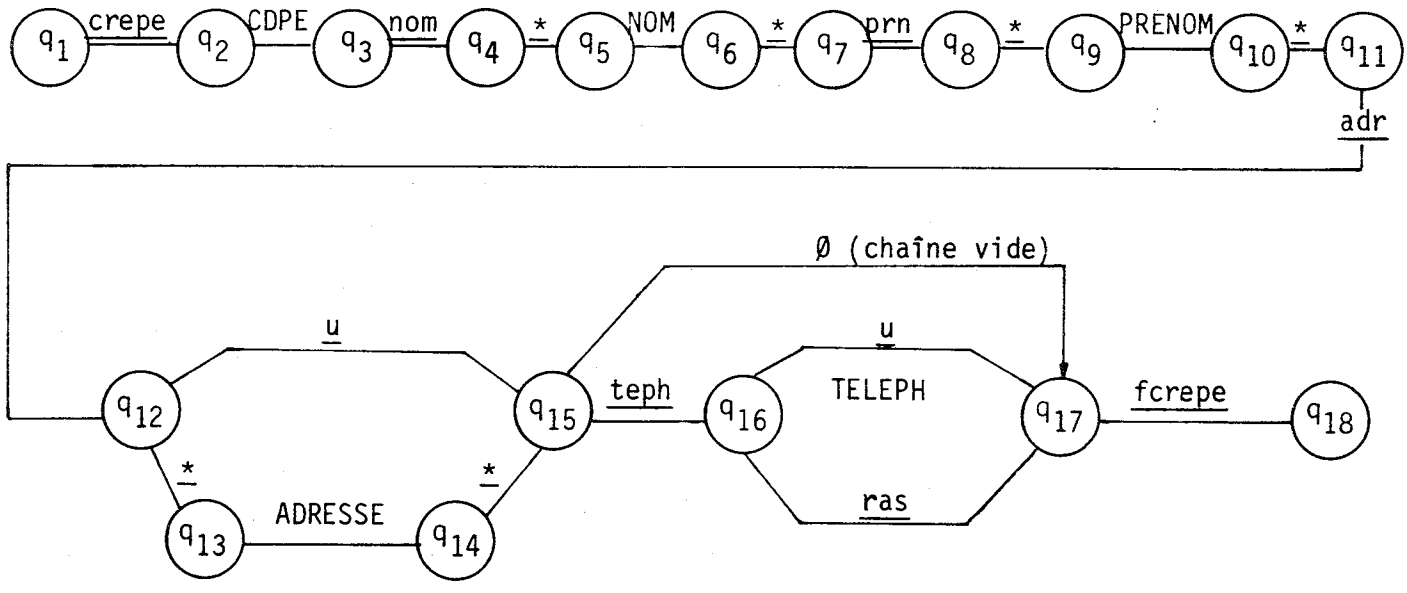
Opération / Entité	Création	Mise à jour	Modification
Groupe	CRESE	MAJGR	MODGR
Personne	CREPE		MODPER
Module		MAJMO	MODMO
Commande		MAJCO	MODCO
Action		MAJAC	MODAC
ENS-REL	CREENS CREREL		MODENSREL
PROPRIETE		MAJPP	MODPP
STRSOCASS	CREST		MODIST
OPERATEUR	CREOP	MAJOP	MODOP

TABLEAU DES NOMS DES DIFFERENTES REGLES DE DESCRIPTION

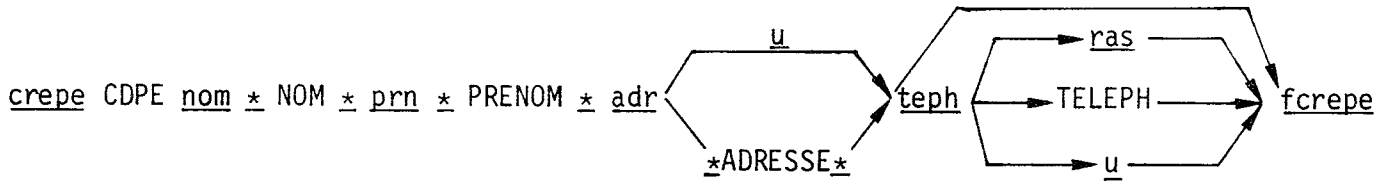
Nous expliquerons ultérieurement pourquoi toutes les cases de ce tableau ne sont pas remplies. Pour l'instant, il est indispensable de commencer par un exemple de présentation d'une RD ; ensuite nous dégagerons les principes généraux de construction des RD.

b) Exemple : Automate correspondant à la RD : "CREPE" (création d'une personne).

b1) Cet automate est représenté par le graphe suivant où l'état initial est q_1 et l'état final q_{18}.



b2) Cette représentation où figure explicitement tous les états est très lourde et peut être avantageusement remplacée par le graphe équivalent suivant :



b3) A cette RD doivent être associés des commentaires explicatifs :

= Toute personne concernée par un projet informatique doit être décrite avec la RD de création d'une personne associée au mot-clé "crepe"

= La description comportera ensuite :

. Le code de la personne CDPE
. Le nom de cette personne, entouré de deux délimiteurs '*', et annoncé par le mot-clé "nom"
. Ensuite, le prénom, limité à gauche et à droite par deux étoiles "*", est précédé du mot-clé "prn"
. Après, le mot-clé "adr" indique que l'on va donner l'adresse de cette personne. Si elle est inconnue de l'analyste au moment où il utilise la RD, celui-ci fera suivre le mot-clé "adr" de la valeur "u" qui signifie : "Undefined" (valeur indéfinie) ou "Ultérieurement précisée". Si l'adresse est connue, elle sera spécifiée alors par un texte entouré de deux délimiteurs "*" .
. Enfin, on peut indiquer, à la suite du mot-clé "teph", le numéro de téléphone où l'on peut joindre cette personne. S'il est provisoirement inconnu, la même valeur "u" sera utilisée. Si, par contre, l'analyste sait explicitement que la personne ne peut être jointe au téléphone, il l'indiquera par la valeur "ras" qui signifie : "Rien A Signaler" (l'information ne peut être définie).
. Si l'analyste ne veut rien indiquer concernant le téléphone, il peut directement signaler que la description de la personne est terminée avec le mot réservé "fcrepe". Ce mot indique la fin de la RD correspondant à CREPE.

Examinons quelques cas de description utilisant cette règle correctement :

crepe FRPE nom * Peccoud * prn * françois gérard * adr * 131 Cours de la libération 38100 Grenoble * teph u fcrepe

crepe CLDE nom * Delobel * prn * claude * adr u teph ras fcrepe

crepe LAMI nom * Larcher * prn * Michel * adr u fcrepe

Il est intéressant, d'autre part, d'examiner quelques exemples d'utilisation incorrecte de la RD :

= crepe ALCH nom * de Chelminski * prn * alfred * teph ras fcrepe

- Nature de l'erreur : même si l'adresse est inconnue, le mot réservé adr est obligatoire.

= crepe STAN prn * andrée * nom * Stiers * adr u teph 87-45-61 fcrepe

- Nature de l'erreur : le nom doit précéder le prénom dans la RD.

= crepe BLPI nom * Blangero prn * Pierre adr u teph 446129-p-336 fcrepe

- Nature de l'erreur : il manque un séparateur '*' à la fin du nom.

Avec cet exemple, nous espérons avoir permis une prise de connaissance expérimentale des mécanismes généraux de construction d'une RD.

c) Principes de construction de l'automate d'état fini correspondant à une RD

La structure de l'automate d'état fini correspondant à une RD doit être construite en respectant certaines normes qui sont :

.) L'état initial de l'automate est suivi d'un seul arc auquel est associé un mot-clé qui est le nom de la RD. Ce mot-clé est toujours suivi du code discriminant permettant d'identifier la réalisation particulière de l'entité sur laquelle opère la RD.

.) Il y a un seul état final dans une RD. Un seul arc y parvient auquel est associé un mot-clé indiquant la fin de la RD.

.) A tout arc est associé : soit un mot-clé (écrit en minuscule) ; soit la valeur d'une des caractéristiques (son code est en majuscule) appartenant ou bien à l'entité sur laquelle opère la RD, ou bien à une autre entité en relation avec celle-ci ; soit trois valeurs réservées qui sont "*", "ras" et "u" ;

certains arcs peuvent être associés à la chaîne vide.

.) Le mot "ras" indique que la valeur d'une caractéristique ne peut être définie.

.) Le mot "u" indique que la valeur peut être définie mais ne peut l'être qu'ultérieurement par l'analyste.

.) La caractéristique à laquelle correspond une valeur est annoncée par le mot-clé qui la précède.

.) Le signe "*" sert de séparateur pour délimiter les valeurs correspondant à des caractéristiques de type texte ou certaines de type mot.

2.5.2. Règle de description : mise à jour d'un module MAJMO

L'analyse descendante sous forme de modules d'une application étant un des choix fondamentaux de MACSI1, il est important de présenter selon quelle règle de description la structure d'un module peut être décrite.

D'après le principe d'analyse descendante, rappelons que :

- l'application est, par définition, un module ayant pour code le nom de l'application
- tout module X ne peut être décomposé en ses éléments que pour autant qu'il ait été lui-même défini précédemment comme élément d'un module Y qui l'englobe et, qu'à ce titre, une réalisation de l'entité module ait été générée ayant un CDMO = X.

Pour ce module X, la description de sa structure est donc une mise à jour notée "majmo".

Le graphe correspondant à la RD "majmo" est présenté en figure 9. Pour faciliter sa présentation, nous avons numéroté les arcs du graphe afin d'y faire facilement référence au cours des commentaires suivants.

2.5.2.1. Commentaires

La RD commence par le mot-clé qui l'identifie "majmo". Il est suivi par le code du module CDMO qui permet de vérifier que ce module a bien déjà été généré. On donne donc ensuite sa définition DEFMOD aussi complète que possible.

Ensuite, la description d'un module commence toujours par la description de son premier élément d'entrée, introduit par le mot-clé "entrée". Mais la description d'un élément, qu'il soit celui en entrée, ou tout élément considéré comme suivant d'un autre élément, se fait toujours de la même manière.

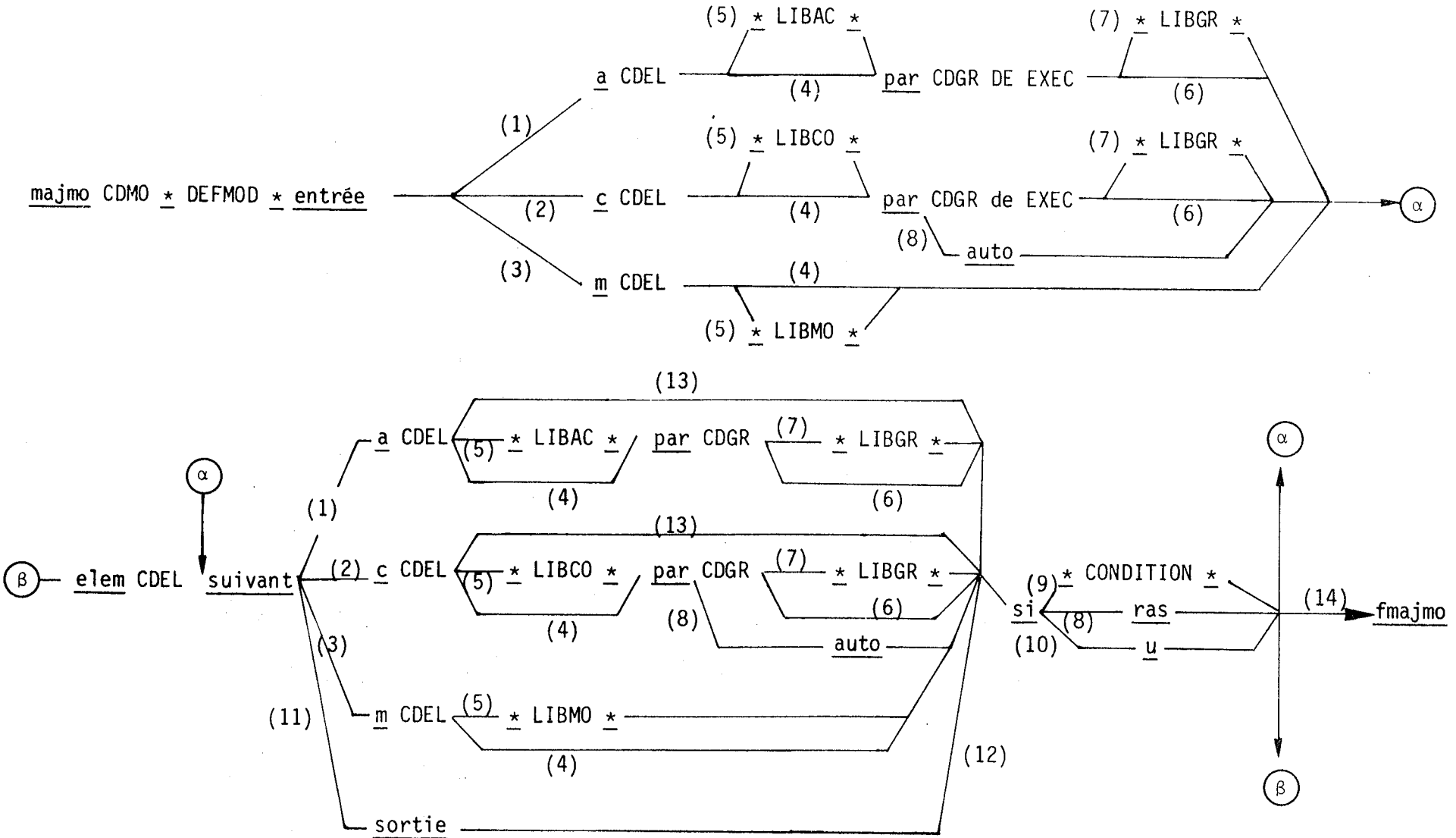


FIGURE 9 - Règle de description MAJMO pour la mise à jour d'un module

Déclaration d'un élément d'un module :

Selon sa nature, son CDEL est un code d'action CDAC (1), un code de commande CDCO (2) ou un code de module CDMO (3). Les trois mots-clé "a" pour action, "c" pour commande ou "m" pour module permettent de préciser cette nature de l'élément.

Si cet élément existe déjà dans la documentation de MACSI1, il suffit simplement de rappeler son code (4). Par contre, dans le cas où il s'agit de la déclaration d'un nouvel élément (elle correspond à la création dans la base documentaire de l'action, de la commande ou du module correspondant), on donnera son libellé (5).

Quand l'élément correspond à une action ou une commande, on indiquera avec la clause "par" le code (CDGR) du groupe qui l'exécute (EXEC). Si le groupe existe déjà, il est inutile de rappeler son libellé (6). S'il s'agit d'un nouveau groupe d'exécution, on devra donner immédiatement son libellé LIBGR (7).

Quand une commande sera exécutée automatiquement, c'est-à-dire que son activation ne dépend pas de l'initiative d'un groupe d'exécution, on indiquera le fait par le mot réservé 'auto' (8).

Définition des éléments-suites d'un élément :

Quand un élément a été déclaré, il faut préciser ses éléments-suites. Il faut d'abord rappeler, après le mot-clé "elem", le code de cet élément, puis chaque élément-suite est introduit par le mot-clé "suivant" et déclaré lui-même selon les règles ci-dessus. Les conditions de son déclanchement sont introduites par le mot réservé "si". Au moment où on décrit cet élément-suite, trois possibilités peuvent se présenter :

1 - Nous savons déjà qu'il n'y a aucune condition particulière à signaler (8). Nous le spécifierons explicitement par le mot réservé "ras".

2 - Nous connaissons précisément ces conditions qui doivent être mentionnées (9). Le texte qui les exprime est indiqué entre deux "*".

3 - Nous ne connaissons pas précisément ces conditions et nous reportons à une mise à jour ultérieure leurs spécifications (10). Nous indiquons par le mot réservé "u" l'absence provisoire de spécifications.

Cas particuliers :

1 - Quand une des suites d'un élément est la sortie du module, on l'indique par le mot réservé "sortie" (11). Les conditions dans lesquelles on sort du module après cet élément doivent toujours être précisées (12).

2 - Quand un élément Y est annoncé comme suivant d'un élément X et qu'il a déjà été décrit précédemment dans le module comme étant aussi le suivant d'un élément Z (cf. règles 5 et 6 au § 2.3.3.), il est inutile de répéter quel est son groupe d'exécution (13) puisque cela a déjà été fait et que le groupe d'exécution associé à une commande ou une action doit être unique à l'intérieur d'un module.

Fin de la RD :

Pour pouvoir terminer la description du module avec le mot réservé "fmajmo" (14), il faut avoir donné pour chaque élément déclaré la liste de tous ses suivants.

Pour être correcte,il faut que la description du module comprenne au moins un point de sortie.

2.5.2.2. Exemple

Nous allons reprendre celui de la gestion des congrès, présenté au § 2.3.3. pour illustrer l'utilisation de cette RD "majmo".

```
majmo M1     * gestion d'un grand congrès (au moins plus de 1000 personnes)
               avec tenue sur ordinateur du fichier des inscriptions
               permettant de faire d'une part une relance pour les inscrip-
               tions irrégulières, d'autre part une gestion de la comptabi-
               lité, enfin l'édition de listes des congressistes inscrits
               (listes triées selon divers critères comme la nationalité,
               la spécialité, l'ordre alphabétique des noms) *
          entrée m M2 * définition du programme scientifique du congrès *
                 suivant m M3 * appel aux communications * si * un an à
                 l'avance *
          elem M3 suivant a A1 * Fixation définitive du programme * par
                 RESPON * comité responsable scientifique du congrès *
                 si ras
          elem A1 suivant a A2 * Edition du programme * par TECH * cellule
                 d'appui technique au congrès * si ras
                  suivant a A3 * Conception de la fiche d'inscription *
                 par TECH si u
          elem A3 suivant a A4 * Tirage de la fiche d'inscription * par
                 IMP * imprimeur * si ras
          elem A4 suivant a A5 * Envoi fiche plus programme * par TECH si *
                 au plus tard 9 mois avant la date d'ouverture du congrès *
          elem A2 suivant a A5 si ras
          elem A5 suivant m M4 * enregistrement des bulletins d'inscriptions
                 remplis * si * six mois avant le congrès ou dès que 50
                 inscriptions sont déjà de retour *
          elem M4 suivant m M4 si * toutes les deux semaines et jusqu'à 8
                 jours de l'ouverture du congrès *
                  suivant c C1 * Edition des inscriptions reçues par listes
                 triées * par auto si * deux jours avant le congrès *
          elem C1 suivant m JOUROU  * Procédures de la journée d'ouverture
                 si * au jour de l'ouverture *
          elem JOUROU suivant m BILCONGR * bilan du congrès * si u suivant
                 c DERLIST * Edition des listes pour tous les participants *
                 par RESPON si u
          elem BILCONGR suivant sortie si * tous les comptes sont soldés *
          elem DERLIST suivant sortie si ras
fmajmo
```

2.6. MISE EN OEUVRE DES REGLES DE DESCRIPTION

Nous ne pouvons pas donner ici le graphe, les commentaires et des exemples d'utilisation correspondant à chaque RD disponible dans MACSI1, comme cela serait fait dans un manuel de référence de la méthode : il faudrait y consacrer un nombre de pages considérable qui n'apporterait pas pour autant au lecteur une connaissance plus précise des principes essentiels de MACSI1. S'il le souhaite, il trouvera la description commentée d'un certain nombre de ces règles dans [7].

Par contre, il est indispensable de définir maintenant selon quelle progression, dans MACSI1, peuvent se faire les spécifications d'un projet rédigées selon ces règles de description.

Plus généralement, il faut indiquer comment, dans la phase d'analyse fonctionnelle d'un projet, doit se dérouler l'activité des analystes. Or, pour exprimer l'ordre suivant lequel ils doivent progresser, il est possible d'utiliser la règle MAJMO. En effet, MACSI1, en tant que méthode assistée par ordinateur, est une application informatique : à ce titre, elle doit donc être, elle-même, descriptible selon le processus de décomposition modulaire proposé avec MAJMO.

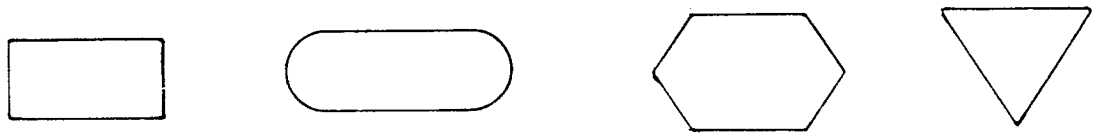
Ce faisant, nous avons l'occasion de donner au lecteur un exemple réel de mise en oeuvre de cette règle de description qui lui permettra de juger par lui-même les possibilités et limites concernant :

- l'analyse descendante d'un projet,
- la précision des concepts tels que ceux d'action, de commande et de groupe d'exécution,
- certaines règles méthodologiques de description d'un module comme l'unicité de l'entrée, les conditions de sortie, l'expression des boucles.

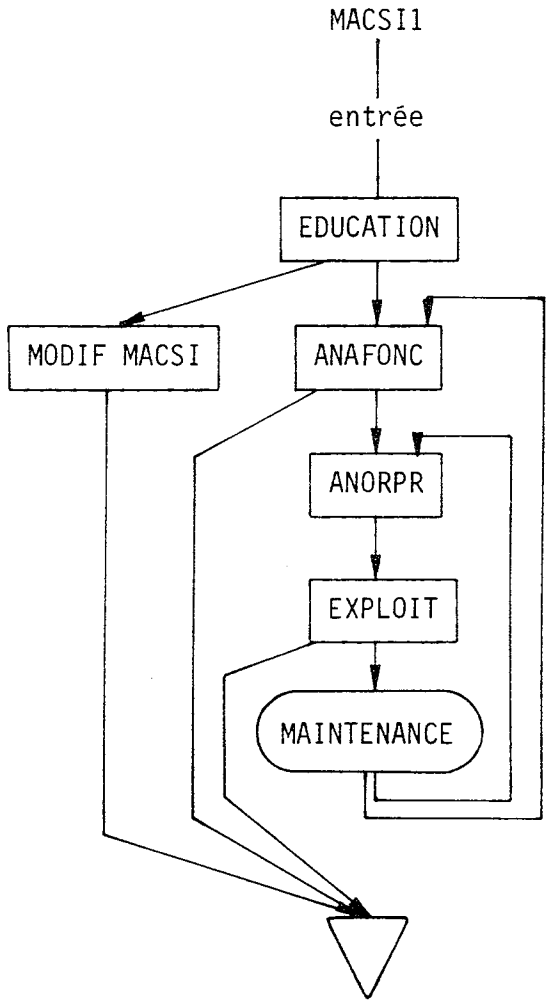
2.6.1. Description modulaire de MACSI1

Nous avons constaté expérimentalement que certains analystes manifestaient le besoin d'une description graphique d'un module préalable à l'utilisation de la RD "MAJMO".

Aussi, nous avons associé à la description de chaque module ci-après une représentation graphique dans laquelle les symboles utilisés ont les significations suivantes :



module action commande sortie

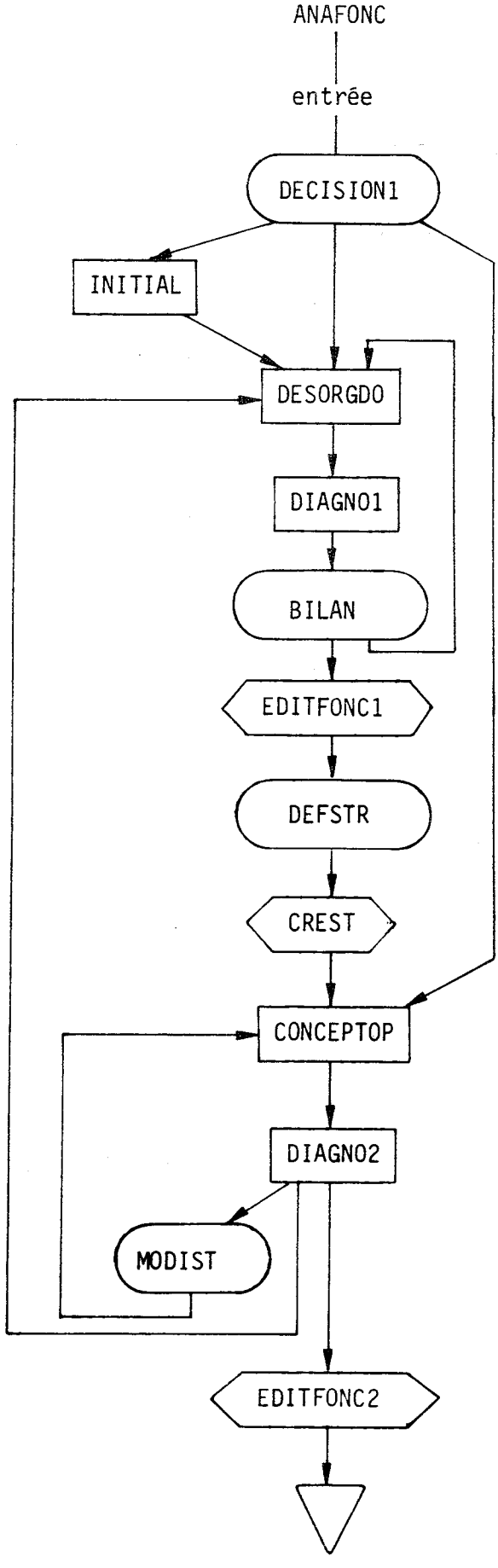


majmo MACSI1 * méthode d'analyse assistée par ordinateur, MACSI1 a pour objectifs essentiels :

- une distinction précise entre analyse fonctionnelle et analyse organique
- la possibilité donnée à certains usagers futurs du projet de participer à ses spécifications sans obligation préalable d'une formation à l'informatique
- la possibilité d'être modifiable pour tenir compte des particularités importantes constatées dans chaque projet *

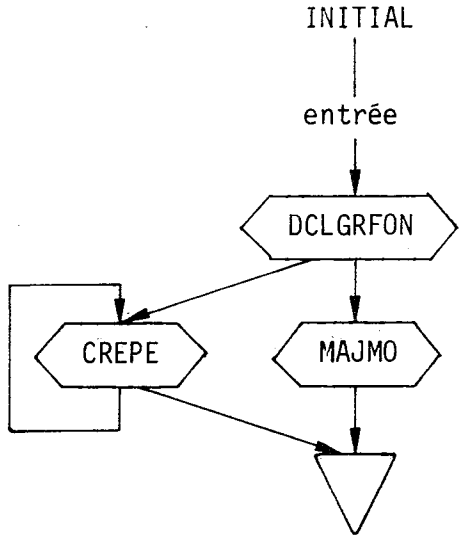
entrée m EDUCATION * formation des analystes devant mettre en oeuvre MACSI1 *

```
        suivant m MODIFMACSI * modification de la méthode *
                    si * elle ne convient pas totalement dans le
                         cadre du projet étudié *
        suivant m ANAFONC * analyse fonctionnelle de l'appli-
                         cation * si ras
elem MODIFMACSI suivant sortie si ras
elem ANAFONC
            suivant m ANAORPR * analyse organique, programmation
                              et tests * si ras
            suivant sortie
                    si * les résultats de l'analyse fonctionnelle
                         ne permettent pas d'envisager une
                         réalisation conforme aux objectifs
                         dans des délais et à un coût supporta-
                         bles ; arrêt du projet *
elem ANAORPR
            suivant m EXPLOIT * exploitation de l'application *
                    si * dès que le lancement peut être effectué *
elem EXPLOIT
            suivant a MAINTENANCE * décision de maintenance de
                                    l'application *
                    par DIRPRO * groupe directeur du projet *
                    si * quand il y a des demandes de modifi-
                         cation du projet *
            suivant sortie si * en cas d'arrêt du projet pour
                              cause de maintenance impossible *
elem MAINTENANCE
            suivant m ANAFONC si * dans tous les cas où il faut
                                   modifier fonctionnellement
                                   les spécifications du projet *
            suivant m ANAORPR si * dans les cas où il ne faut faire
                                   que des modifications mineures
                                   de programmation qui ne
                                   changent en rien les spécifi-
                                   cations fonctionnelles de
                                   l'application *
fmajmo
```

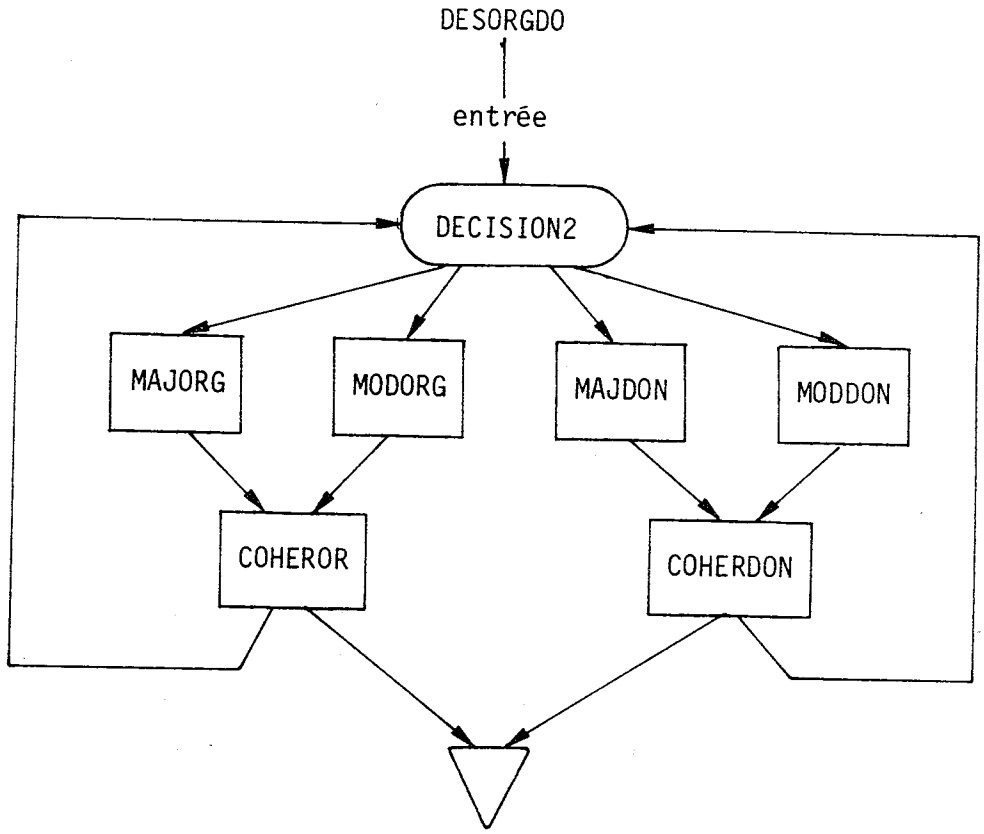

ANAFONC
entrée
DECISION1
INITIAL
DESORGDO
DIAGNO1
BILAN
EDITFONC1
DEFSTR
CREST
CONCEPTOP
DIAGNO2
MODIST
EDITFONC2

```
majmo   ANAFONC * analyse fonctionnelle du projet. Elle concerne d'abord :
                    - La description de l'organisation (modules, actions,
                      groupes d'exécution, services)
                    - La description des données (ensembles,propriétés,
                      relations)
                    - La description des traitements (commandes).
                  Après des bilans de cohérence et l'édition d'un dossier
                  de synthèse, les informaticiens établissent une structure
                  Socrate associée à la description sémantique des données
                  et écrivent les opérateurs correspondant aux commandes *
                entrée a DECISION1 * s'agit-il du début d'une analyse ou de la
                                     suite d'un projet déjà commencé *
                                     par CHPRO * chef de projet *
                        suivant m INITIAL * initialisation de l'analyse *
                                  si * en cas de démarrage de l'analyse du
                                       projet *
                        suivant m DESORGDO * description de l'organisation
                                             et des données *
                                  si * en cas de prolongation d'une analyse
                                       déjà commencée *
                        suivant m CONCEPTOP * description des opérateurs *
                                  si * en cas de modification portant
                                       uniquement sur les opérateurs *
                elem INITIAL suivant m DESORGDO si ras
                elem DESORGDO suivant m DIAGNO1 * diagnostics globaux sur les
                                                  spécifications concernant
                                                  l'organisation et les données *
                                                  si ras
                elem DIAGNO1 suivant a BILAN * bilans sur la cohérence des
                                               spécifications concernant les
                                               données et l'organisation *
                                               par CHPRO si ras
                elem BILAN   suivant m DESORGDO si * les spécifications sont
                                                     insuffisantes *
                             suivant c EDITFONC1  * édition du dossier exhaustif
                                                    des données * par CHPRO
                                                    si ras
```

```
elem EDITFONC1  suivant a DEFSTR * définition de la structure
                                   Socrate associée *
                         par COORINF * coordinateur des travaux
                                       informatiques *
                         si * la description de l'organisation
                              et des données est jugée suffisante *
elem DEFSTR suivant c CREST * chargement du source de la
                               structure Socrate associée dans
                               la base documentaire * par COORINF
                         si * après contrôle par le chef de
                              projet de la cohérence entre le
                              modèle sémantique de données et la
                              structure Socrate (cf. § 2.3.6.2.) *
elem CREST suivant m CONCEPTOP si ras
elem CONCEPTOP suivant m DIAGNO2 * diagnostics sur les opérateurs *
                                   si ras
elem DIAGNO2 suivant m DESORGDO
                         si * la description sémantique des données
                              est remise en cause *
              suivant c MODIST * modification, par l'éditeur
                                 Socrate, de la structure associée *
                           par COORINF
                         si * la structure associée est remise en
                              cause sans modification préalable
                              de la description sémantique des
                              données *
              suivant c EDITFONC2  * édition dossier analyse
                                     fonctionnel complet *
                           par CHPRO
                         si * le chef de projet juge l'ensemble
                              des spécifications correctes *
elem MODIST suivant CONCEPTOP si ras
elem EDITFONC2 suivant sortie si ras
fmajmo
```




```
majmo   INITIAL * Le démarrage de l'analyse d'un projet avec MACSI1 doit
                  commencer par une identification des différents groupes
                  fonctionnels et de leurs membres. Ensuite, il faut donner,
                  à partir du module ayant le nom du projet, une première
                  décomposition de celui-ci *
            entrée c DCLGRFON * déclaration des groupes fonctionnels *
                   par CHPRO
                       suivant c MAJMO * mise à jour de module *
                           par CHPRO
                               si * pour le module correspondant à l'appli-
                                    cation *
                       suivant c CREPE * pour une personne : nom, prénom,
                                         adresse, téléphone *
                           par CHPRO
                               si * pour chaque membre des groupes fonc-
                                    tionnels du projet *
            elem CREPE suivant c CREPE
                               si * pour toutes les personnes des groupes
                                    fonctionnels *
                       suivant sortie
                               si * quand tous les membres des groupes fonc-
                                    tionnels sont identifiés *
            elem MAJMO suivant sortie si ras
fmajmo
```



majmo DESORGDO

* La description fonctionnelle de l'organisation et des données est réalisée par :

- la décomposition des modules en actions et commandes avec leurs groupes d'exécution correspondants selon un processus d'analyse descendante,
- l'identification des ensembles de base, de leurs relations et des propriétés qui leur sont associées.

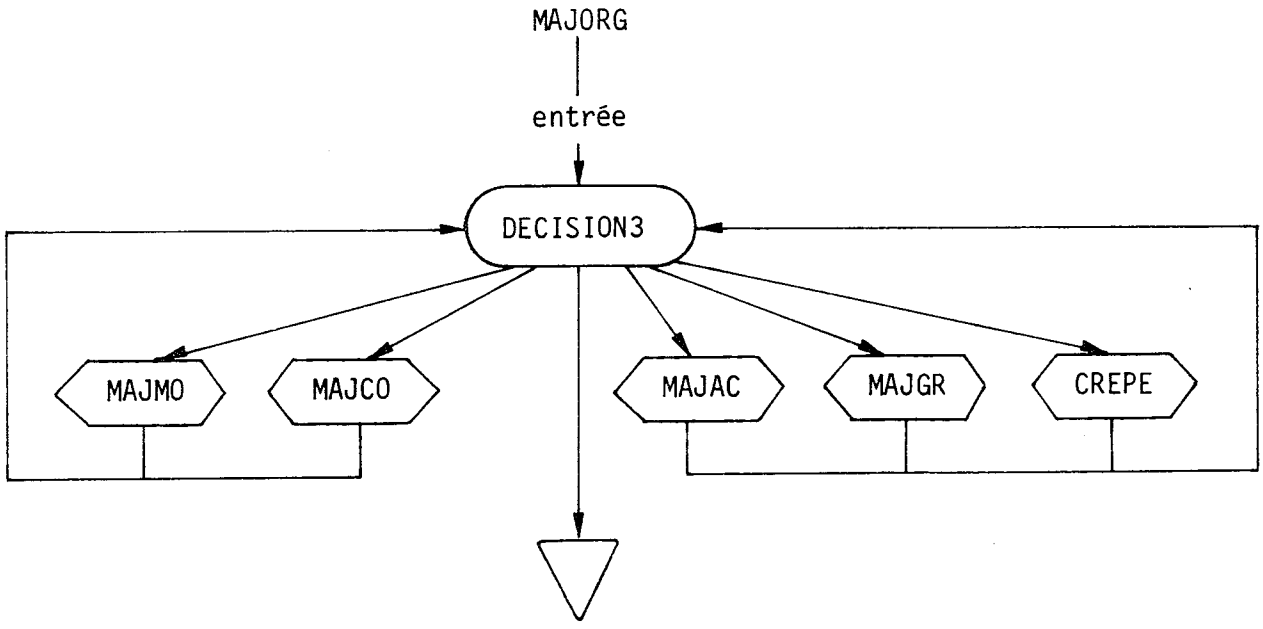
Il existe trois modalités de description d'un composant :

- la création d'un composant quand on identifie son code, son libellé et ses propriétés,
- la mise à jour d'un composant quand on identifie ses propriétés alors que son code et son libellé ont déjà été déclarés dans la description d'un autre composant avec lequel il est en relation,
- la modification d'un composant quand on change les valeurs de certaines de ses propriétés ou que l'on supprime le composant.

*

```
entrée a DECISION2 * décision du type de mise à jour ou modi-
                     fication à faire *
        par GRORDO * groupe fonctionnel chargé de concevoir
                     l'organisation et les données *
          suivant m MAJORG * mise à jour de l'organisation *
                  si * il faut créer ou mettre à jour de
                       nouveaux modules, actions, commandes,
                       groupes d'exécution, personnes *
          suivant m MODORG * modification de l'organisation *
                  si * il faut modifier les caractéristiques
                       existantes de modules, actions, commandes,
                       groupes ou personnes qui s'avèrent
                       inadéquates *
          suivant m MAJDON * mise à jour de la descript. séman-
                             tique des données *
                  si * il faut compléter la descr. sémantique *
          suivant m MODDON * modification de la description
                             sémantique des données *
                  si * il faut modifier la description sé-
                       mantique existante *
elem MAJORG
          suivant m COHEROR * contrôles automatiques de cohérence
                              de l'organisation décrite * si ras
elem MODORG
          suivant m COHEROR si ras
elem MAJDON
          suivant m COHERDON * contrôles automatiques de
                               cohérence de la description
                               des données * si ras
elem MODDON
          suivant m COHERDON si ras
elem COHEROR
          suivant a DECISION2
                  si * les analystes veulent corriger immé-
                       diatement les incohérences détectées,
                       s'il y en a *
```

```
                    suivant sortie
                            si * il n'y a pas d'erreurs détectées par
                                 COHEROR ou bien, s'il y en a, les
                                 analystes veulent aussi faire les
                                 diagnostics de DIAGNO1 avant de faire
                                 de nouvelles spécifications *
            elem COHERDON
                    suivant a DECISION2
                            si * les analystes veulent corriger immé-
                                 diatement les incohérences détectées
                                 dans la description des données *
                    suivant sortie
                            si * les analystes veulent exécuter les
                                 tests supplémentaires de DIAGNO1
                                 avant de modifier ou compléter les
                                 spécifications existantes *
fmajmo
```



majmo MAJORG * mise à jour des éléments suivants :

- modules déclarés comme composants d'un autre module (majmo)
- commandes ou actions déclarées comme composants d'un module (majco, majac)
- groupes d'exécution identifiés dans des décompositions de modules (majgr)
- personnes identifiées comme devant faire partie de groupes d'exécution (crepe)

*

entrée a DECISION3 * choix des mises à jour restant à faire * par ANORG * personnes chargées de la conception de l'organisation : elles font partie de GRORDO *

suivant c MAJMO par ANORG

si * il reste des modules à décomposer *

suivant c MAJCO * définition précise d'une commande * par ANORG

si * concernant les commandes pas encore définies et déclarées comme éléments de modules *

suivant c MAJAC * pour une action : définition et type * par ANORG

si * concernant les actions déclarées comme éléments de modules et encore indéfinies *

```
                    suivant c MAJGR * pour un groupe d'exécution : sa
                                      fonction et éventuellement certains
                                      de ses membres * par ANORG
                            si * concernant les groupes identifiés dans
                                 les modules et encore indéfinis *
                    suivant c CREPE par ANORG
                            si * concernant les personnes identifiées
                                 comme membres d'un groupe d'exécution *
                    suivant sortie
                            si * les analystes veulent évaluer la
                                 cohérence des spécifications qu'ils ont
                                 écrites *
          elem MAJMO suivant a DECISION3 si ras
          elem MAJCO suivant a DECISION3 si ras
          elem MAJAC suivant a DECISION3 si ras
          elem MAJGR suivant a DECISION3 si ras
          elem CREPE suivant a DECISION3 si ras
fmajmo
```

```
majmo MAJDON * description de la logique des données. Elle fait appel à
          trois commandes :
            - CREENS = identification des ensembles de base
            - CREREL = création de relations entre des ensembles de base
                       ou entre d'autres relations
            - MAJPP  = mise à jour des propriétés d'ensembles de base
                       ou de relations
        *
         entrée a DECISION4 * choix des définitions restant à faire *
                 par ANDO * groupe responsable de la description des
                            données : il fait partie de GRORDO *
                   suivant c CREENS * définition d'un ensemble de base :
                                      code, définition, taille, liste
                                      des codes de ses propriétés *
                 par ANDO si u
```

```
                    suivant c CREREL * spécification d'une relation :
                                       code, définition, arguments et
                                       fonction d'accès, liste des codes
                                       des propriétés * par ANDO si u
                    suivant c MAJPP * mise à jour, pour chaque propriété,
                                      de son type et de sa décomposition *
                                      par ANDO
                              si * pour les propriétés déclarées avec
                                   CREENS ou CREREL *
                    suivant sortie si * les analystes veulent faire des
                                        tests de cohérence sur la logique
                                        des données *
          elem CREENS suivant a DECISION4 si ras
          elem CREREL suivant a DECISION4 si ras
          elem MAJPP suivant a DECISION4 si ras
fmajmo
```

Arrêtons ici la progression de la décomposition modulaire de MACSI1. La structure des modules MODORG et MODDON est analogue à celles de MAJORG et MAJDON : les commandes qu'elles font apparaître sont alors, non plus des commandes de création ou de mise à jour, mais des commandes de modification.

2.6.2. Bilan sur l'utilisation des règles de description

La règle MAJMO présentée et utilisée dans les paragraphes précédents est celle qui, parmi les RD, a la complexité de loin la plus grande. Le lecteur aura pu - du moins nous l'espérons - se faire une idée personnelle sur les possibilités d'utilisation de cette règle (et des concepts de module, action, commande et groupe d'exécution qui la sous-tendent) par des analystes n'ayant pas de compétence particulière en informatique. Les premières expériences de formation que nous avons conduites, nous laissent bien augurer des possibilités d'assimilation de ce genre de formalisme par des analystes ayant une culture limitée en mathématiques et informatique.

2.7. PROGRAMMES D'AIDE A L'ANALYSE POUR LE CONTROLE DES SPECIFICATIONS ET LEUR DIFFUSION

Nous avions indiqué que MACSI1 comportait des programmes d'aide à l'analyse assurant soit des tests de cohérence sur les spécifications produites, soit une gestion sélective de la documentation.

Dans le paragraphe précédent, sont apparus un certain nombre de modules ou de commandes dont le libellé laisse deviner qu'ils assurent ces fonctions de contrôle ou de documentation automatique.
Ce sont :

- m COHEROR = contrôles automatiques de cohérence de l'organisation
- m COHERDON = contrôles automatiques de cohérence de la description des données
- m DIAGNO1 = diagnostics globaux sur les spécifications concernant l'organisation et les données
- m DIAGNO2 = diagnostics concernant les opérateurs.

Par ailleurs, nous avons fait figurer deux commandes d'édition de la base documentaire :

- c EDITFONC1 = édition du dossier exhaustif des données
- c EDITFONC2 = édition du dossier d'analyse fonctionnelle complet.

Il ne peut être envisagé de donner la liste exhaustive de tous les contrôles et de tous les programmes d'édition réalisés ou possibles : elle serait longue et fastidieuse. Nous préférons dégager ici une typologie des contrôles et des éditions en donnant, dans chaque cas, quelques exemples.

2.7.1. Types de programmes de contrôle

On peut distinguer, dans les programmes de contrôle, différents niveaux.

A) Tests de cohérence ponctuels : ce sont tous ceux qui portent sur les spécifications, concernant un seul composant, stockées dans une seule réalisation d'une entité de la base documentaire. La plupart de ces contrôles sont effectués par les programmes qui assurent l'exécution de chaque règle de description par mise à jour de la base documentaire. Examinons la nature des contrôles qu'ils assurent.

a1 - Le respect de la syntaxe des RD :

Exemples : Vérifier dans la description d'un module avec "majmo" que :

- chaque action ou commande a un groupe d'exécution (sauf s'il s'agit d'une boucle : cf. les arcs numérotés (13) sur la figure 9)
- la condition, exprimée par le mot-clé "si", figure pour tout élément
- les libellés figurent entre deux "*"
- les modules n'ont pas de groupe d'exécution.

Vérifier dans la description d'une personne (par la RD : CREPE) que son adresse est bien donnée (même si c'est avec la valeur "u") puisqu'elle est obligatoire.

a2 - Le respect de la sémantique de chaque RD explicitée par les commentaires associés à l'automate.

Exemple : Pour un module, vérifier :

- il existe au moins un élément ayant pour suite "sortie",
- chaque élément a au moins un suivant,
- chaque élément, qui figure déjà dans le dictionnaire, n'est pas indiqué avec un autre libellé (arcs numérotés (4) et (6) sur la figure 9),
- chaque nouvel élément a, par contre, son libellé (arcs numérotés (5) et (7) sur la figure 9).

a3 - Respect de l'obligation d'utiliser des codes discriminants

Toute déclaration d'un nouveau composant, avec un code qui est déjà affecté à un composant d'un autre type, doit être refusée.

B) Tests de cohérence globaux : ce sont tous ceux qui nécessitent, pour être exécutés, un accès en lecture à la totalité de la base documentaire.

b1 - Inventaire des spécifications incomplètes

Il faut pouvoir, à tout moment, faire un inventaire, soit des caractéristiques ayant la valeur "u", soit des éléments déclarés qui ne sont pas encore mis à jour.

Exemple : - liste des modules, actions ou commandes déclarés comme éléments d'un module qui ne sont pas encore mis à jour par majmo, majac, majco.

- liste des propriétés déclarées, non encore définies par majpp.

- liste des propriétés composées dont les composantes ne sont pas définies.

b2 - Diagnostics méthodologiques sur l'ensemble des spécifications

Ils permettent de donner certaines présomptions d'anomalies dans la conception. Ces diagnostics exploitent, en général, les relations qui ont été posées dans le modèle de MACSI1 entre ses divers types de constituants. Ils donnent des bilans qui ne sont pas des constats certains d'erreur : ils sont fournis pour alerter le chef de projet sur certaines possibilités d'anomalies. Voyons quelques exemples :

* Liste des actions, commandes ou modules qui apparaissent comme éléments dans plusieurs modules.

Intérêt : vérifier, pour chacun, qu'il assure exactement la même fonction dans ses différentes occurrences.

* Liste des groupes d'exécution dont les membres sont rattachés à différents services administratifs.

Intérêt : il peut y avoir conflit d'attribution entre ces services au moment où il faudra mettre en place le groupe.

* Liste, pour chaque groupe d'exécution, de l'ensemble des tâches (commandes et actions) dont il a la charge.

Intérêt : cette liste facilite les contrôles suivants :

- Est-ce que l'ensemble de ces tâches confiées au groupe est homogène ?

- N'y a-t-il pas, parmi ces tâches, deux ou plusieurs d'entre elles qui ont des noms différents mais qui, en fait, sont identiques ?

* Liste des commandes de sortie qui ne sont suivies d'aucune action de type 1 ou 3.

Intérêt : est-ce que la finalité de cette commande est clairement analysée ?

* Liste des commandes d'entrée qui ne sont précédées d'aucune action de type 2 ou 4.

Intérêt : est-ce que l'origine des données prises en compte par cette commande est clairement exprimée ?

* Liste des ensembles X qui ont une seule relation(avec un autre ensemble Y).

Intérêt : est-ce que X n'est pas une propriété de Y ?

2.7.2. Types de programmes documentaires

Il est possible de même de classer les programmes de documentation selon l'usage qui est fait de la documentation qu'ils produisent. Nous distinguerons quatre types d'utilisations de la documentation :

a) Gestion des nomenclatures : Il est important d'éditer à intervalles réguliers le glossaire,qui est la duplication du contenu de l'entité DICO. Il permet aux analystes d'éviter de donner des codes déjà utilisés à de nouveaux composants. Ce glossaire peut prendre différentes formes selon l'ordre d'édition.

b) Dossier partiel permettant de récapituler les seules informations utiles à une tâche d'analyse précise.

Par exemple, il est nécessaire d'éditer la liste de tous les ensembles, leurs relations et leurs propriétés,pour préparer la définition de la structure Socrate associée. C'est le rôle de la commande EDITFONC1 avant l'action DEFSTR.

De même, il faut faire un inventaire de toutes les commandes pour pouvoir étudier quels sont les opérateurs de service qu'elles exigent.

c) Evaluation des conséquences d'une modification dans les spécifications.

Il s'agit, dans l'hypothèse d'une modification d'un élément, d'obtenir la liste de tous les éléments du système d'information, en relation avec celui-là, dont les caractéristiques seraient aussi à modifier.

Exemple : S'il faut changer la définition d'une propriété, quelle est la liste de tous les opérateurs concernés ? On exploitera les caractéristiques ESPDONNEE, OPERFILS et OPERPERE dans l'entité OPERATEUR.

Exemple : Une opération, considérée initialement comme une action, doit être, en fait, automatisée et considérée comme une commande : la liste de tous les modules dont cette action est un élément est nécessaire.

d) Edition du dossier d'analyse exhaustif.

Il est, enfin, bien évident qu'il faut procéder à une édition complète du contenu du fichier documentaire lorsque l'analyse est considérée comme terminée, c'est-à-dire lorsque les contrôles automatiques précédents ne détectent plus d'anomalies et que le chef de projet réceptionne les spécifications. C'est là le rôle de la commande EDITFONC2 à la fin de l'analyse fonctionnelle.

2.8. FLEXIBILITE DE MACSI1 : SES POSSIBILITES D'ADAPTATION

Comme nous l'avons indiqué au chapitre 1, la présentation qui vient d'être faite de MACSI1 porte uniquement sur les concepts essentiels du modèle et la structure des règles de description d'un projet. La base documentaire proposée au § 2.4. n'est, en fait, qu'un sous-ensemble de celle réellement mise en oeuvre dans MACSI1. Nous rappelons au lecteur qu'il trouvera dans le travail de A. de CHELMINSKI [7] toutes les caractéristiques du projet MACSI1 qui ont été passées sous silence ici, ainsi que certains aspects concernant son implémentation.

Notre présentation, volontairement simplifiée, a eu pour objectif de suggérer, par touches successives, les possibilités d'extension de MACSI1, témoignant ainsi d'un effort prioritaire pour dégager, moins une méthode d'analyse, qu'une méthode de construction de méthodes d'analyse.

Pour conclure, il peut être intéressant de situer les extensions envisageables, d'une part quant à leurs finalités et, par ailleurs, quant aux techniques qui les rendent possibles.

2.8.1. Objectifs des extensions

Examinons, parmi les extensions possibles, celles qui sont réalisées mais n'ont pas été présentées ici, ou celles que dégage l'utilisation expérimentale de MACSI1.

a) Etendre le modèle pour pouvoir documenter complètement et contrôler les spécifications réalisées pendant la phase d'analyse organique et de programmation (cf. [7] chapitre 3) : introduire les notions de fichier, document, enregistrement de fichier, unité de traitement, etc ....

b) Enregistrer systématiquement les dates de prise en compte et les états d'avancement des spécifications (cf. la caractéristique ETAT de COMMANDE), pour tous les composants, de façon à permettre un suivi efficace des délais de réalisation de l'analyse d'un projet.

c) Affiner le modèle de description de l'organisation en ajoutant, en particulier, aux ACTIONS et COMMANDES d'autres caractéristiques qui permettraient de préciser plus complètement la fonction de ces composants.

d) Essayer de compléter le modèle pour permettre d'évaluer, entre les phases d'analyse fonctionnelle et organique, le coût d'exploitation du projet informatique, compte tenu de caractéristiques portant sur l'évaluation des volumes d'activité.

2.8.2. Conditions techniques de réalisation de ces extensions

Examinons, par ailleurs, quelles sont les options techniques dans la démarche de construction de MACSI1 qui doivent être conservées pour pouvoir réaliser ces extensions facilement.

a) Il est possible de faire certaines modifications de la structure Socrate qui sont très facilement réalisables :

- Modification d'une caractéristique, du type "liste de valeurs", en ajoutant des valeurs supplémentaires.

- Ajout d'une ou plusieurs caractéristiques à l'intérieur d'une entité de plus haut niveau déjà existante.

- Ajout de nouvelles relations entre deux entités de plus haut niveau existantes.

- Ajout de nouvelles entités de plus haut niveau, correspondant dans le modèle à de nouveaux types de composants de base d'un système d'information.

Toutes ces formes de modifications sont réalisables et donneront une structure documentaire nouvelle sur laquelle, une fois qu'elle sera compilée, fonctionneront tous les programmes déjà écrits et opérationnels, sans qu'il soit besoin de les modifier.

b) Il faut pouvoir modifier les règles de description aussi facilement que la structure documentaire.

Toutes les RD étant un ensemble d'automates d'états finis déterministes, il est possible dans MACSI1 de stocker les descriptions de ces automates comme des paramètres, qui dirigent un programme général

d'analyse syntaxique (cf. [7] § 4.e.4.). Aussi, pour chaque nouvelle RD, doivent être écrits uniquement les seuls sous-programmes correspondant aux contrôles sémantiques du type de ceux présentés ci-dessus au § 2.7.1. alinéa (a2).

c) Il faut, enfin, souligner que le choix du système de base de données Socrate se confirme comme essentiel,pour permettre des modifications rapidement implémentées. Au niveau de programmes simples, comme les programmes de contrôle ou d'édition, un ingénieur connaissant bien le langage peut écrire et mettre au point environ 400 instructions par semaine. Certes, l'expérience prouve que les durées de traitement avec MACSI1 ne sont pas très bonnes du fait des performances de Socrate, mais c'est le tribut nécessaire payé pour avoir la flexibilité indispensable que procure le choix d'un logiciel de bases de données.

2.8.3. Conclusion

La finalité de toute extension doit d'abord être étudiée par rapport aux possibilités des analystes qui s'en serviront et à l'intérêt qu'ils y trouveront : il ne servira à rien de concevoir des modifications, très élaborées conceptuellement, du modèle de description d'un projet informatique si elles ne sont pas utilisables pratiquement par les hommes auxquels on les propose. Certains lecteurs penseront, à l'issue de ce chapitre, que le modèle proposé est trop simpliste ; d'autres le trouveront trop compliqué ; d'autres, enfin, inadapté : qu'importe finalement qu'ils disposent de moyens efficaces pour le modifier si, par ailleurs, ils oublient qu'une méthode d'analyse est faite pour des analystes. Le terme générique d' "analyste" est finalement bien ambiguë ... presque autant que celui d' "utilisateur" : à chacun de le préciser.

0

0 0

CHAPITRE 3

MACSI2, UNE EXTENSION DE MACSI1 POUR CONSTRUIRE LE PROTOTYPE D'UNE APPLICATION INFORMATIQUE

3.1. DIFFICULTES POUR CONSTRUIRE UNE APPLICATION INFORMATIQUE ADAPTEE A SES UTILISATEURS

Avec MACSI1, nous avons essayé de disposer d'une documentation gérée par ordinateur permettant de vérifier la cohérence d'un projet selon certains critères. Il est possible de tester certains aspects de faisabilité du projet. L'importance et la diversité de ces contrôles en vue d'aider les analystes, dépend, comme nous l'avons vu, des extensions apportées au modèle de description.

Cependant, il apparaît expérimentalement que, pour bon nombre de projets informatiques, quels que soient le nombre et la diversité des contrôles effectués, ils seront souvent insuffisants pour garantir le succès du projet.

Il existe, en effet, dans certains centres informatiques, des projets dont l'engineering n'a pas souffert d'un manque de spécifications, qui ont été correctement contrôlés et suivis en délai et en coût et qui, cependant, sont considérés finalement comme des échecs.

Quelle est la cause de ces constats d'échec ?

Essentiellement, les utilisateurs d'un projet, c'est-à-dire les personnes concernées dans leur activité quotidienne par son fonctionnement, le jugent inadapté à leurs besoins et à leurs contraintes de travail lorsqu'ils ont à s'en servir effectivement.

Ce constat n'est pas nouveau : il a été développé dans d'innombrables articles de la grande presse et de la presse spécialisée depuis quelques années.

Plusieurs solutions ont été proposées pour pallier cette carence d'adaptation des projets informatiques aux desiderata de leurs utilisateurs. Certaines mesures d'améliorations ont été mises en oeuvre :

a) La formation de ces utilisateurs à l'informatique par un recyclage plus ou moins intensif. L'idée de cette formation procède de l'hypothèse qu'en connaissant mieux l'outil informatique, les utilisateurs sauront mieux apprécier les implications de leurs demandes de traitements.

b) La formation des informaticiens aux techniques de gestion, en espérant qu'elle les rende plus sensibles aux préoccupations des gestionnaires.

c) Quelquefois, la cellule "Etudes" d'un service central informatique d'une entreprise a été dissoute parce qu'elle était considérée comme un ghetto susceptible de provoquer une certaine "schizophrénie technique". Les chefs de projets, analystes et programmeurs ont été dispersés dans les diverses directions opérationnelles de l'entreprise pour avoir un contact quotidien avec l'activité des services.

Toutes ces mesures, pour intéressantes qu'elles soient, ne nous semblent pas, à l'usage, avoir répondu aux espérances de ceux qui les ont prises. Elles n'ont pas toujours permis de construire des projets informatiques enfin bien adaptés aux besoins de leurs usagers. Elles posent, en effet, comme condition sine qua non de réussite d'un projet, la nécessité de donner la responsabilité des spécifications fonctionnelles du projet à ses futurs utilisateurs.

Il existe un consensus pour juger que, si dans la conception d'un produit informatique, les utilisateurs ont été correctement formés aux techniques informatiques et disposent,en face d'eux,de la collaboration éclairée d'informaticiens largement initiés aux méthodes de gestion, ces futurs utilisateurs doivent être capables d'exprimer,complètement et clairement,leurs besoins en information.

Mais,ces utilisateurs sont des personnes qui, au même moment, peuvent aussi envisager l'acquisition d'une voiture ou d'un appartement et qui doivent, dans cette perspective, savoir quelles sont les caractéristiques essentielles qu'elles exigent d'un logement ou d'un moyen de locomotion.

Comment se comportent-elles dans de telles situations de choix ? Pour les mieux formées d'entre elles, elles sauront lire les plans d'un architecte mais elles ne se contenteront pas de les étudier avant de prendre l'engagement financier d'achat : elles préfèreront, avant toute décision irréversible, visiter longuement l'appartement témoin pour "se faire une idée" ! De façon analogue, rien ne vaut un essai sérieux sur route, en étant soi-même au volant, pour évaluer des caractéristiques d'une voiture qui ont leur importance dans le choix d'un véhicule :

niveau de bruit, place disponible pour les jambes et les bagages, tenue de route, confort des sièges, etc ... etc .... Chacun de nous, avec ses propres critères d'utilité et ses normes personnelles de satisfaction, a besoin, pour faire un choix qu'il juge sérieux, de voir, de toucher, d'utiliser un produit analogue à celui qu'il envisage d'acquérir.

Nous souhaitons, par cette comparaison, illustrer notre conviction personnelle que tout individu est ainsi fait qu'il ne peut souvent pas formuler des recommandations précises sur les caractéristiques d'un produit qu'il envisage d'acquérir s'il ne bénéficie pas, au préalable, de l'expérience pratique obtenue par l'utilisation d'un produit analogue. Cette expérience concrète permet de formuler des critiques vis-à-vis du produit essayé, qui deviennent des recommandations pour le produit dont l'acquisition est souhaitée.

Même en informatique, il apparaît souvent que les utilisateurs ne peuvent pas avoir des idées précises de leurs besoins tant qu'ils n'ont pas fait l'expérience d'une application analogue à celle dont ils sont chargés de produire le cahier des charges.

Cette nécessité d'avoir, dans certains cas, un prototype expérimental d'une application informatique de gestion avant de figer définitivement les spécifications de la version opérationnelle, nous a été confirmée par diverses constatations : certaines peuvent être rappeler ici pour concrétiser les remarques précédentes.

a) Pour certains projets informatiques dans le domaine des systèmes d'exploitation, la construction d'un prototype apporte un lot de renseignements très intéressants sur les possibilités du système qu'il est utile de développer et celles qu'il est dangereux de laisser aux utilisateurs.

Ainsi, ayant eu l'occasion de suivre d'assez près le développement du projet Socrate, nous avons été très frappés de l'importance donnée par son responsable, J.R. ABRIAL, aux délais nécessaires pour que toutes sortes de personnes se servent intensivement de chaque version expérimentale du produit et puissent ainsi exprimer des critiques motivées. Ces remarques ont compté de façon déterminante dans l'évolution de Socrate : par exemple, la possibilité initialement offerte de définir conditionnellement des caractéristiques dans une structure de données a été supprimée au vu de l'usage qui en était fait.

b) En évoquant avec M. FORGE, Directeur du CXP (Centre d'eXpérimentation des Packages), les motivations des entreprises qui se portent acquéreurs de packages, il nous a fait part d'une remarque assez surprenante a priori :

Certaines entreprises achètent un produit-programme non pas pour l'utiliser de façon régulière mais simplement pour savoir, après un usage intensif, quelles sont les caractéristiques du produit qui ne conviennent pas ou manquent, compte tenu des besoins spécifiques de l'entreprise. Ayant pu ainsi faire un inventaire exhaustif des qualités et défauts du produit-programme, elles savent alors précisément définir les caractéristiques propres du produit qu'elles doivent construire pour leur usage interne.

A partir de ces réflexions, il a donc été envisagé de développer, dans le cadre du projet MACSI, une extension MACSI2 de MACSI1 permettant de donner une réponse aux questions suivantes :

- que doit-être le prototype d'une application informatique de gestion ?
- quels sont les enjeux de la réalisation d'un prototype et comment préciser les cas où elle peut être conseillée ?
- quelles sont les conditions de faisabilité d'un prototype ?
- enfin, est-il possible, sans solution de continuité, de définir une extension de MACSI1 facilitant la réalisation d'un prototype ?

3.2. DEFINITION : PROTOTYPE D'UNE APPLICATION INFORMATIQUE

Il faut entendre par "prototype" un modèle de la structure fonctionnelle de l'application. Nous utilisons le terme de MODELE au sens de MINSKY cité par Mélèze dans [26] :

"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".

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

- Pour un 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 l'application -

Essayons donc de faire l'inventaire des questions précises que se pose quelqu'un qui participe à la conception d'un projet informatique sachant que, plus tard, il en sera un des principaux utilisateurs.

1 - A quoi est-ce que cela va servir ? Ou, plus précisément, quelles informations seront produites et pour quelles tâches de gestion seront-elles utilisées ?

2 - Quelles données devront être fournies par les services pour que l'application fonctionne ? Quelles modifications doivent être introduites dans leurs procédures pour que l'exactitude de ces données soit suffisante ? Quelle charge de travail supplémentaire cela donnera-t-il aux services ? Ceux-ci ont-ils une compétence suffisante pour assurer ces procédures administratives ?

3 - Le coût prévu de développement de l'application justifie-t-il les avantages nouveaux procurés ?

Il apparait donc évident que, pour répondre à ces questions, le prototype est une autre application informatique qui doit être fonctionnellement équivalente à celle étudiée. 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 dans la réalisation du prototype.

- 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.

Plus complètement, on peut exprimer par le tableau suivant les similitudes et différences existant entre une application et son prototype.

CONCERNANT DANS L'APPLICATION	ON PEUT ETUDIER ET REALISER DANS LE PROTOTYPE (équivalence fonctionnelle)	IL RESTE CEPENDANT A PRECISER POUR L'APPLICATION FINALE (différences organiques)
(1) LES FICHIERS	- Toutes les relations logiques entre les informations contenues dans les fichiers. - Tester la définition de ces informations en construisant des jeux d'essais.	- La structure physique des fichiers : support, type d'accès (séquentiel, indexé, direct), facteur de blocage, etc .. - Les programmes standards de transformation de cette structure (tri, fusion, éclatement, etc ...) - La constitution initiale des fichiers permanents.
(2) LES PROCEDURES DE SAISIE ET DE CONTROLE DES DONNEES, EN ENTREE DE L'APPLICATION	Implémentation rapide de leur logique en écrivant avec un logiciel, de mise en oeuvre facile, toutes les procédures associées aux différentes créations, mise à jour ou suppression de valeurs ainsi que tous les contrôles associés à ces opérations.	- Les meilleures solutions techniques pour la collecte des données (cartes perforées, saisie multi-clavier, console on-line ou remote batch, lecture optique, etc ..) compte tenu du volume des flux d'entrée. - Le regroupement des contrôles en programmes de contrôles suivant le regroupement des entrées dans différents fichiers mouvements.
(3) LA PRODUCTION D'INFORMATIONS POUR LES GESTIONNAIRES, EN SORTIE DE L'APPLICATION	Implémentation rapide de leur logique en écrivant avec un logiciel, de mise en oeuvre facile, toutes les procédures associées aux différentes opérations de calcul et d'édition des résultats.	- La périodicité, le volume, le support (listing, écran cathodique, télétype, microfilm, etc ...) associé à chaque sortie. - Un regroupement efficace de la logique exprimée par les procédures du prototype dans des programmes écrits avec d'autres logiciels pour obtenir des performances d'exécution meilleures.
(4) SES ASPECTS ERGONOMIQUES 4.1 - Difficultés éprouvées par le personnel pour fournir l'information en entrée 4.2 - Intérêt pour les utilisateurs de recevoir telle ou telle information.	4.1 - Exécution, par les personnes chargées effectivement de la saisie, des procédures implémentées au point (2) - Etude de leur comportement - Modification rapide du prototype en fonction de leurs remarques. 4.2 - Maniement, par les personnes utilisatrices de l'application, des procédures écrites au point (3) - Etude de leurs critiques (procédures jugées inutiles, incomplètes ou souhaitables bien que n'étant pas écrites) - Modification rapide du prototype, compte tenu de ces critiques.	4.1 - Les circuits administratifs de collecte de l'information, les dessins définitifs des imprimés, les variations de volume des flux de documents. 4.2 - Les circuits de diffusion des résultats.

EQUIVALENCE FONCTIONNELLE ET DIFFERENCES ORGANIQUES

entre une application et son prototype

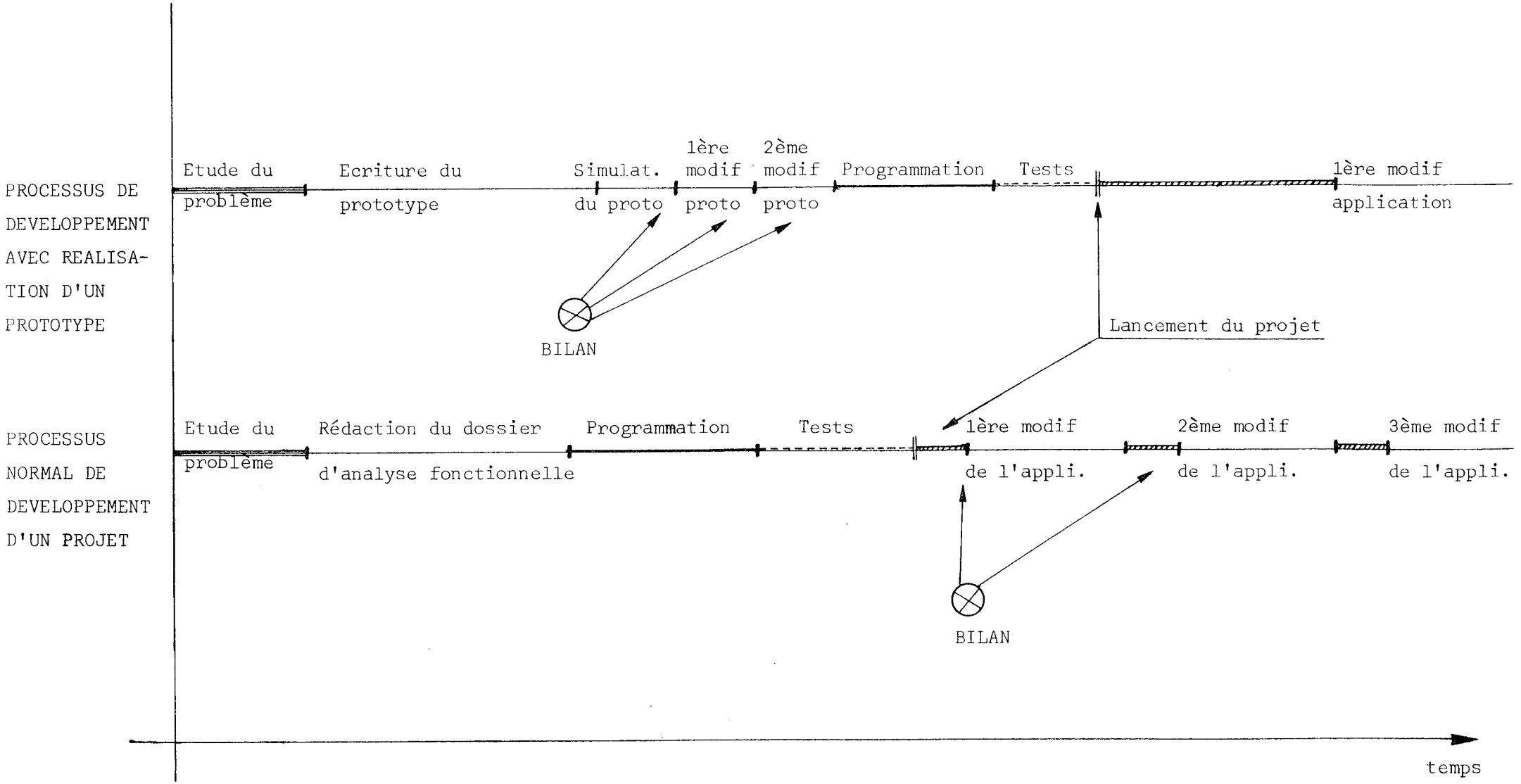
3.3. LA REALISATION D'UN PROTOTYPE : UN AUTRE SCHEMA DE DEVELOPPEMENT D'UN PROJET

Dans le processus normal de développement d'un projet, à la fin de la phase d'analyse fonctionnelle, il doit y avoir théoriquement un accord précis entre utilisateurs et techniciens informatiques sur la structure de l'application. Mais, en fait, la programmation de l'application, les tests des programmes et les modifications nécessaires des procédures administratives une fois réalisés, c'est réellement au moment du lancement du projet que va être évaluée son efficacité : souvent des modifications dans la conception, parfois importantes, vont être exigées rapidement par les services utilisateurs à la lumière des premiers mois d'utilisation de cette application. En général, ces demandes de modifications seront d'autant plus importantes que les transformations introduites par l'application informatique dans les méthodes de travail de ces services sont plus profondes.

Si ces modifications sont trop considérables, le coût de leur réalisation peut représenter une partie importante du coût initial d'étude et de développement du projet. Elles seront surtout considérées par les services qui les demandent comme un échec dans l'analyse du projet.

Aussi, dans ce genre de situation, la réalisation d'un prototype de l'application a-t-elle, pour enjeu fondamental de permettre un nouveau processus de développement du projet dans lequel puissent s'exprimer, avant la réalisation de la version opérationnelle, toutes les critiques qu'appellent sa conception. Ces critiques sont suggérées par l'utilisation intensive du prototype. Après examen de leur contenu, le prototype est modifié ; il fait à nouveau l'objet de critiques après utilisation. Ainsi, se prolonge, par approximations successives, la mise au point jusqu'à ce que les services qui testent le prototype 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 dans le prototype les conduira à ne pas demander des modifications majeures dans cette version opérationnelle [17].

Le schéma suivant résume, en référence à une même base de temps, les différentes étapes de développement du projet suivant l'un ou l'autre processus de développement retenu.



Légende :

Exploitation opérationnelle.

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

3.4. CONDITIONS DE FAISABILITE D'UN PROTOTYPE

Pour que la construction d'un prototype soit possible, il apparaît que deux conditions principales doivent être remplies :

Condition 1 : La programmation du prototype doit faire appel à des moyens permettant, d'une part sa réalisation dans des délais rapides pour ne pas trop augmenter la durée des études et, d'autre part, des modifications, souvent importantes, très vite implémentées pour que les utilisateurs,qui testent le prototype,n'attendent pas longtemps entre deux versions de ce prototype.

Condition 2 : Pour que le prototype soit un modèle de toute l'application, il doit représenter non seulement ses aspects proprement informatiques, mais aussi l'enchaînement des actions dans les circuits administratifs qui sont inséparables de l'application en amont et en aval. Ces actions expriment, en effet, l'origine des informations saisies dans l'application et l'utilisation qui sera faite de celles produites par l'application.

3.4.1. Pouvoir réaliser et modifier rapidement le prototype

Parmi les moyens actuellement disponibles pour construire ou modifier un prototype dans des délais rapides, des logiciels de base de données, utilisés en respectant certaines normes de programmation peuvent présenter des avantages certains.

Pour ce qui nous concerne, nous avons réalisé la construction de prototypes avec le logiciel Socrate. Il permet à des analystes-programmeurs confirmés ou des ingénieurs-informaticiens une productivité de programmation suffisamment importante pour que le prototype soit réalisé dans des délais comparables à ceux d'une étude spécifiée uniquement par des dossiers d'analyse fonctionnelle correctement rédigés.

Mais pour que les modifications des programmes du prototype soient faciles, il faut adopter un certain nombre de normes dans la construction des programmes qui doivent être respectées. Elles portent à la fois sur la structure des données du prototype et la structure des procédures écrites.

3.4.1.1. Normes pour la structure des données : exemples

a) Pour que la structure des données du prototype soit flexible, il faut que les ensembles et relations de la structure logique des données correspondent à des entités de plus haut niveau. Il faut donc éviter, autant que faire se peut, les imbrications de blocs et d'entités.

b) Les deux fonctions d'accès d'une relations doivent être permises pour ne pas, a priori, privilégier l'accès à un des arguments plutôt qu'à l'autre.

c) Il faut éviter de déclarer des caractéristiques comme liste de valeurs. En effet, il est possible que, pour détecter l'anomalie d'une mise à jour avec une valeur n'appartenant pas à la liste, le programmeur soit tenté de laisser au système Socrate le soin de ce genre de contrôle ; à l'exécution, ce choix s'avère mauvais car le contrôle des entrées n'est pas au niveau de l'application mais au niveau du système Socrate : en cas d'erreur, il est alors impossible de reprendre le contrôle du déroulement du prototype.

3.4.1.2. Normes pour la structure des procédures : exemples

d) La programmation de chaque procédure correspondant à l'entrée de données doit permettre, lorsqu'elle fonctionnera en mode conversationnel, deux genres d'utilisation appelés respectivement le mode "bavard" et le mode "silencieux" :

- Le mode "bavard" fournit à l'utilisateur qui teste le prototype toutes les indications dont il a besoin chaque fois qu'il doit fournir une donnée : la signification de la donnée, la liste de ses valeurs possibles, éventuellement son format. Le mode "bavard" est ainsi un mode de fonctionnement de la procédure qui doit permettre un apprentissage de son fonctionnement au moment de ses premières utilisations.

- Une fois que, par le mode bavard, l'utilisateur a pris connaissance du déroulement d'une procédure, il doit pouvoir, dans un souci de rapidité du dialogue à la console, utiliser cette procédure en mode "silencieux" qui supprime tout commentaire de la machine à chaque demande de données et résume celle-ci à l'édition d'un simple code mnémonique.

Pour avoir toute la liberté d'apprentissage souhaitable, l'utilisateur doit pouvoir, à chaque fois qu'il lui est demandé une donnée, frapper un caractère spécialement réservé (par exemple " / ") à la place de la donnée : ce caractère correspond à une commande qui fait basculer le prototype du mode "bavard" au mode "silencieux" ou inversement et ceci pour toutes les procédures du prototype.

e) Dans un souci de programmation du prototype la plus modulaire possible, il est alors évident que tous les messages nécessaires, soit pour guider l'utilisateur en mode "bavard", soit pour lui expliquer la nature des erreurs détectées sur les données qu'il fournit, doivent être facilement modifiés. Il ne faut donc pas les intégrer, dans les procédures, par une instruction de la forme :

i <chaîne de caractère>

ce qui a pour conséquence l'obligation de compiler à nouveau une procédure chaque fois qu'il est nécessaire de modifier la chaîne de caractère du message.

Il faut, au contraire, créer une entité de messages MESSAGE dont on peut provoquer l'édition par appel d'une procédure EDIT (Y_i) où Y_i est le numéro d'ordre de la chaîne de caractère qu'il faut éditer dans l'entité MESSAGE.

f) De même, par un souci de performance dans les délais de programmation du prototype, il est très conseillé de remplacer une suite de tests, permettant le choix d'activation d'une procédure parmi n, par un appel associatif de cette procédure. Plus précisément, il est souhaitable de remplacer toute séquence du genre suivant :

```
m BUFFER = ext
si BUFFER = 1 alors call P1 fin
si BUFFER = 2 alors call P2 fin
------------------------------
si BUFFER = n alors call PN fin
```

par la séquence suivante :

```
m BUFFER = ext
si existe un PROGRAMME X_i ayant CODE = BUFFER ;
   alors call X_i sinon EDIT (Y_i) fin
```

3.4.2. Pouvoir simuler les actions en amont ou en aval des traitements informatiques

Comme l'usage expérimental du prototype doit permettre aux utilisateurs de juger de la conception du projet, il faut qu'ils évaluent non seulement la logique des traitements sur ordinateur, mais aussi la cohérence des actions qui précèdent toute procédure d'entrée de données ou qui suivent toute procédure de sortie de données calculées. Pour cela, il faut définir une forme de simulation qui permette de représenter les circuits administratifs en aval et en amont des traitements proprement informatiques. Ne pas le faire, c'est reconnaître que ceux-ci peuvent avoir un intérêt et une fiabilité indépendants des mécanismes administratifs pour lesquels ils ont été conçus.

Une modalité de simulation possible peut être l'organisation, sous la conduite du chef de projet, de séances d'évaluation du projet qui, dans leur déroulement, soient analogues aux séances de travail mises en place dans le cadre de certaines sessions de "jeu d'entreprises" : à chaque séance participent toutes les personnes des différents groupes d'exécution chargés d'accomplir le lot d'actions et de commandes qui constitue l'ordre du jour de la séance.

Dans une telle séance de simulation, pourraient être ainsi expliqués, puis évalués :

- Soit un circuit d'acquisition de données, composé d'actions de création, de transmission, de duplication et de contrôle d'un document suivies d'une commande d'entrée de données, elle-même suivie en général d'un circuit de correction des anomalies détectées par la commande.

- Soit un circuit de diffusion de données éditées par ordinateur, en insistant sur les règles de transmission de ces résultats et surtout sur les actions de gestion qu'ils permettent.

Il faut évidemment que le chef de projet respecte, dans l'organisation de ces séances de simulation, l'ordre chronologique des différentes procédures tel qu'il apparaît dans une description modulaire du projet avec MACSI1.

3.4.3. Insuffisances de MACSI1 pour construire un prototype

Quand le chef de projet dispose de l'ensemble des opérateurs du prototype, écrits en respectant les normes de programmation précédentes et qui correspondent aux commandes identifiées dans une application décrite avec MACSI1, il faut constater qu'il ne dispose pas dans MACSI1 des moyens lui permettant de contrôler si, au cours des séances de simulation, les conditions d'enchaînement des actions et commandes, tel que l'exprime la structure de modules, sont correctement respectées. Cette présentation exacte de l'enchaînement dynamique des actions et commandes dans l'ordre où elles doivent être activées est pourtant indispensable pour une bonne compréhension des procédures par ceux qui les exécuteront.

Peut-on envisager, de façon réaliste, que soit laissée une responsabilité totale au chef de projet pour les faire exécuter dans l'ordre où elles sont décrites dans les modules de MACSI1 ? L'expérience prouve, en fait, qu'il est difficile de conduire une animation de séance en faisant respecter cet ordre. Cette difficulté amène, en général, le chef de projet à privilégier uniquement, dans le test du prototype, l'utilisation des commandes, en laissant de côté la mise en oeuvre effective des actions.

Pour aider le chef de projet dans l'utilisation du prototype limité aux seules commandes, il semble possible d'écrire un programme en Socrate qui supervise l'ordre d'appel des différents opérateurs correspondant aux commandes. Mais cette solution est très limitée, car si l'ordre d'exécution des commandes est modifié après une utilisation du prototype, il faut réécrire ce programme superviseur. Or, dans ce superviseur, sont programmées toutes les conditions définies sous forme de texte libre après le mot réservé "si" dans les modules de MACSI1, conditions qui expriment dans quels cas une commande peut ou ne peut pas être activée à la suite d'une autre. Comme le nombre de ces conditions est en général important, l'écriture de ce superviseur d'appel des différents opérateurs est d'une grande complexité : sa maintenance est donc aussi très difficile et ne permet pas une modification facile du prototype.

3.5. PRINCIPES D'EXTENSION DE MACSI1 VERS MACSI2

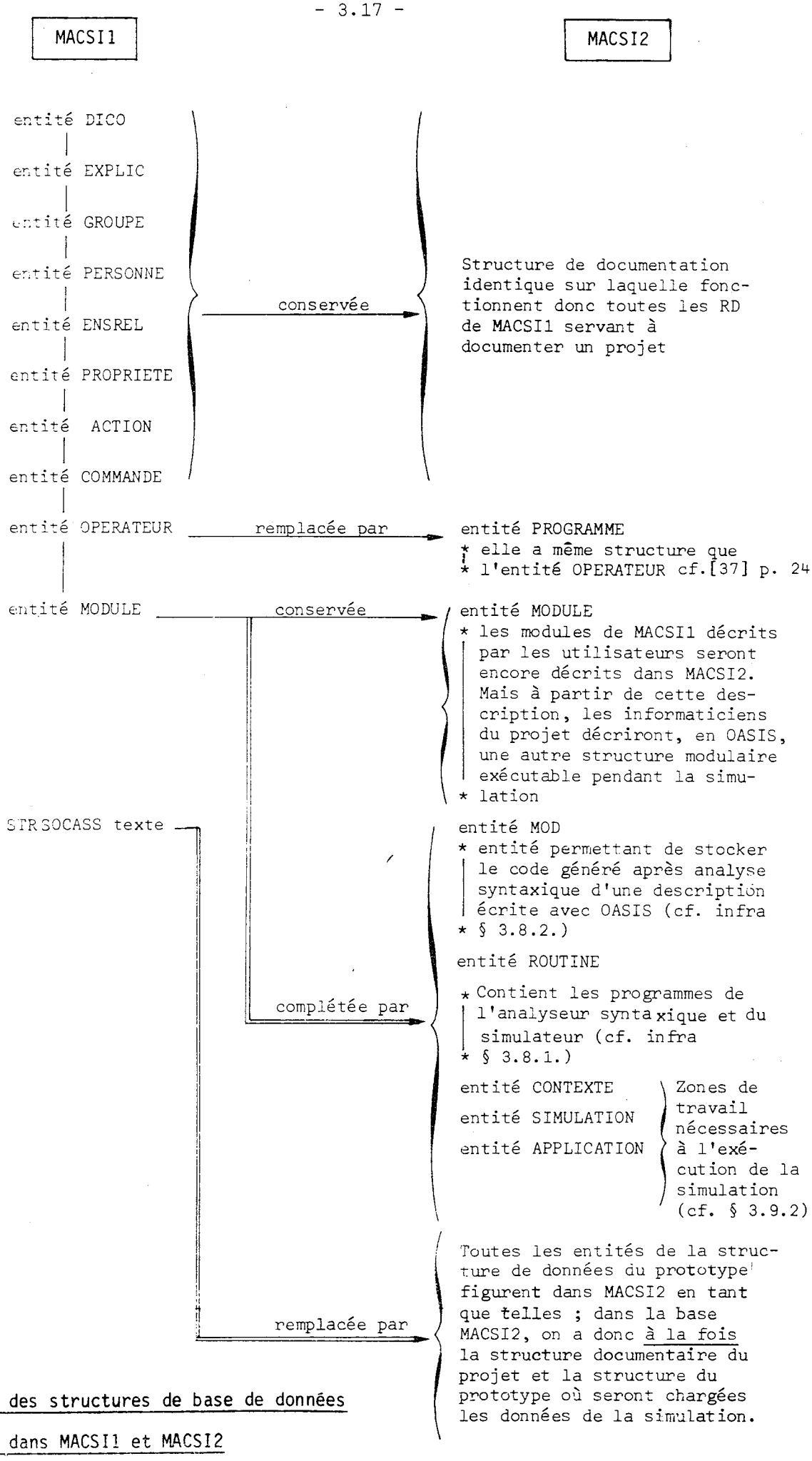
Toutes les difficultés précédentes ont donc pour origine le fait que, dans MACSI1, la spécification des modules autorisée par l'automate MAJMO est purement descriptive et ne permet pas d'exécuter sous le contrôle de la machine la dynamique d'enchaînement des commandes et actions qu'elle exprime.

Aussi, dans MACSI2, il a fallu ajouter à cette formulation des modules de MACSI1, une autre forme de représentation des modules à partir de laquelle il soit possible de contrôler par ordinateur l'utilisation d'un prototype d'une application.

Plus précisément, nous allons présenter un langage qui permette d'exprimer fondamentalement la même réalité que celle décrite par les modules de MACSI1 ; mais la conception de ce langage permet de lui associer un analyseur syntaxique dont le code généré sera exploité par un superviseur chargé de contrôler la simulation des opérateurs et actions d'un prototype.

C'est ce langage, baptisé OASIS (Outil d'Aide pour la SImulation de Systèmes) que nous allons définir dans les pages qui suivent.

MACSI2 se présente alors comme une extension de MACSI1. Les modalités de cette extension sont synthétiquement représentées par le schéma ci-joint qui exprime les différences entre les deux structures de base de données utilisées par ces deux versions du projet MACSI.



Comparaison des structures de base de données utilisées dans MACSI1 et MACSI2

3.6. ELEMENTS DU LANGAGE "OASIS"

L'axiome de la grammaire du langage OASIS est une définition de module.

3.6.1. Définition de "module"

<définition de module> → DEFMOD <code de module><libellé><liste de comp> FINMOD

Une définition de module est composée de :

- un mot-clef "DEFMOD" permettant d'indiquer, dans le système MACSI2, le début d'une description de module avec OASIS
- un nom de module qui doit être un code discriminant permettant de l'identifier
- un texte libre pour le décrire succinctement
- la liste de ses éléments terminée par le mot-clef "FINMOD".

La structure de cette liste permet de représenter l'enchaînement dynamique des éléments constituants le module.

Ces différents éléments s'appellent les COMPOSANTS du MODULE.

Attention : Le prototype de l'application que l'on veut construire est lui-même considéré comme un module. Il est identifié par l'instruction :

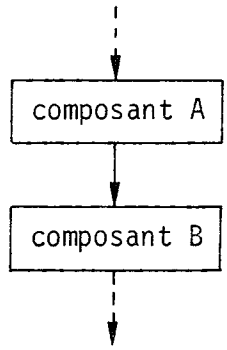
<application> → ap <code application> <libellé> <code du module associé>

Liste de composants

<liste de comp> → <composant> [<composant>]*
<composant> → <composant simple><libellé>|<composant complexe>

Une liste de composants est une suite finie non vide de composants, totalement ordonnée. La relation d'ordre par laquelle un composant A est suivi d'un composant B dans la liste, exprime que l'exécution de B suit séquentiellement celle de A.

Cette règle donne le premier schéma de construction : le schéma séquentiel.



Un composant est soit simple, soit complexe.

3.6.2. Les composants simples

<composant simple> → <action>|<programme>|<module>

Un composant simple est la citation d'un composant de type action, programme ou module. Le type est défini par un mot-clef :

"a" pour un composant action
"p" pour programme
"m" pour module.

Le composant "action"

<action> → a <code action> / groupe d'exécution

Une action est une tâche autre qu'un traitement de l'information automatisé, faisant appel pour être exécutée à des ressources (hommes et moyens) autres que celles du service informatique. La définition d'une action est celle présentée dans MACSI1. A ce titre :

- une action est identifiée par son nom (code action) qui doit être discriminant

- l'utilisateur doit indiquer le nom du groupe chargé de l'exécution de cette action (EXECUTANT2).

Le composant "programme"

<programme> → p <code programme> / <groupe d'exécution> |
p <code programme> / auto

Dans l'hypothèse d'un prototype, l'objet "programme" complète dans MACSI2 à la fois les notions de commande et d'opérateur de MACSI1. Nous avons changé de terminologie pour ne pas créer de confusion et à l'entité OPERATEUR de MACSI1 est substituée l'entité PROGRAMME dans MACSI2. Un programme est une opération de traitement automatisé de l'information. Il est écrit en SOCRATE et travaille sur la structure de données du prototype.

- Tout programme sera associé à une réalisation de l'entité PROGRAMME où l'on retrouvera en particulier son SOURCE (cf. [37] page 24).

- Un programme est identifié par son nom (code programme) qui doit être discriminant.

- De même que pour une action, l'utilisateur doit indiquer le nom du groupe chargé éventuellement d'activer ce programme. Si l'utilisateur veut indiquer que l'activation de ce programme sera faite automatiquement sans intervention externe d'un groupe, il l'indique par "auto"

Le composant "module"

<module> → m <code module>

Tout module X est considéré comme un composant simple dans un module Y, mais la structure de X doit par ailleurs être décrite par une instruction DEFMOD avant ou après la description de Y.

- Un module est un ensemble composé de programmes, d'actions, de modules, de composants complexes

- Il est identifié par un nom, <code module>

- Un module n'a pas de groupe d'exécution car seules les tâches élémentaires (actions, programmes) sont affectuées à des groupes d'exécution.

3.6.3. Les composants complexes

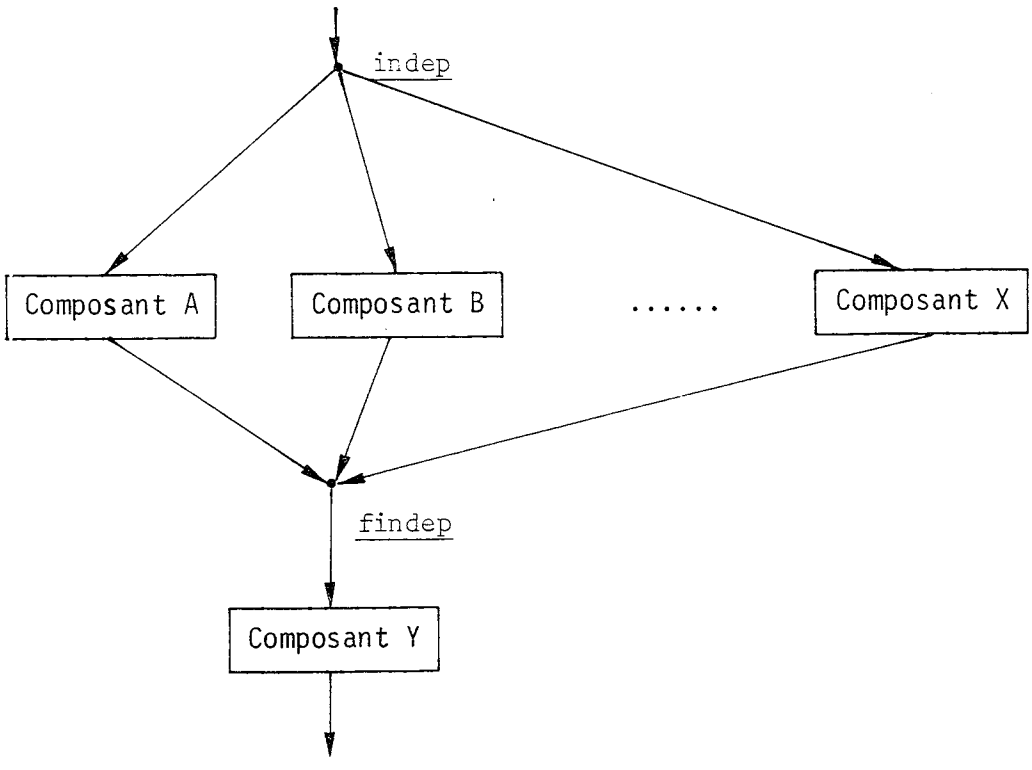
<composant complexe> → <indep>|<si>|<tant que>|<suivant>

Les composants complexes définissent les règles de construction de base permises entre les tâches élémentaires (actions, programmes, ...), la nature de la règle étant définie par un mot-clef : indep, si, tantque, suivant.

Le composant "indépendant"

<indep> → indep <Liste de comp> findep

- Schéma d'indépendance



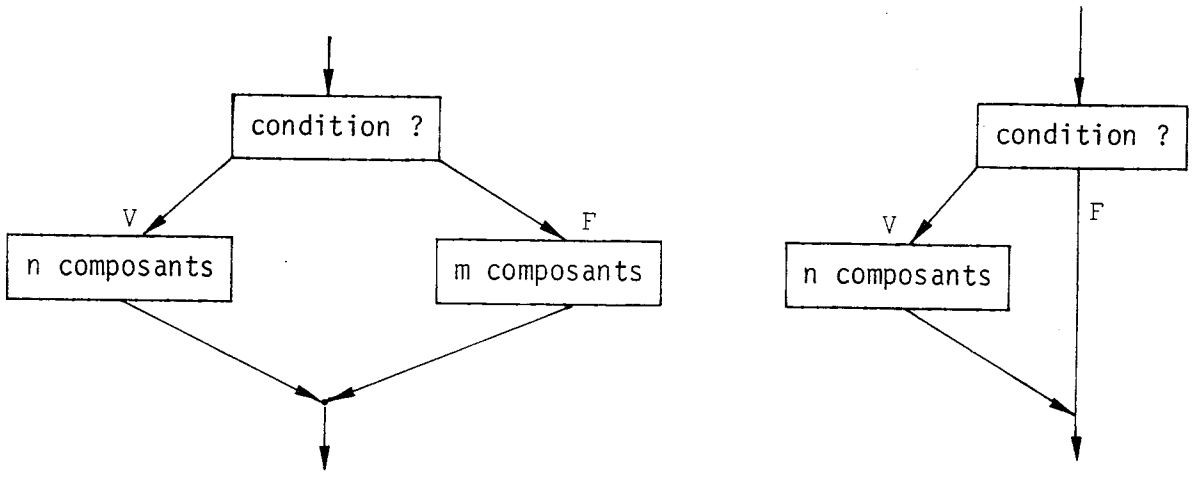
- Un composant d' "indépendance" exprime l'exécution collatérale d'un ensemble de composants A, B, ..., X

- Le composant Y qui suit le composant "indep" ne peut être activé que lorsque A, B, ... et X auront tous été activés et terminés.

Le composant "si"

<si> → si <condition><libellé> alors <liste de comp> [sinon<liste comp>] fsi

- Schémas conditionnels



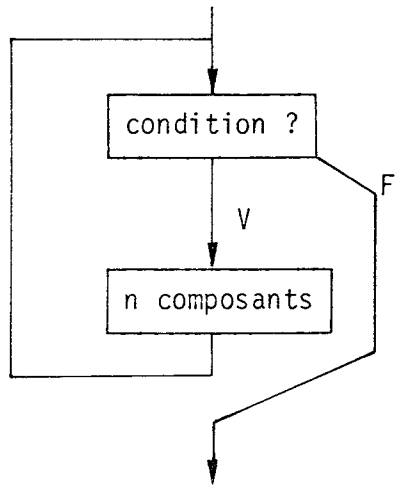
- C'est l'instruction "si" classique que l'on trouve dans tous les langages de programmation.

- L'évaluation de la condition permet de connaître la liste de composants à exécuter.

Le composant "tantque"

<tantque> → tantque <condition> <libellé> faire <liste de comp> ftq

Schéma itératif



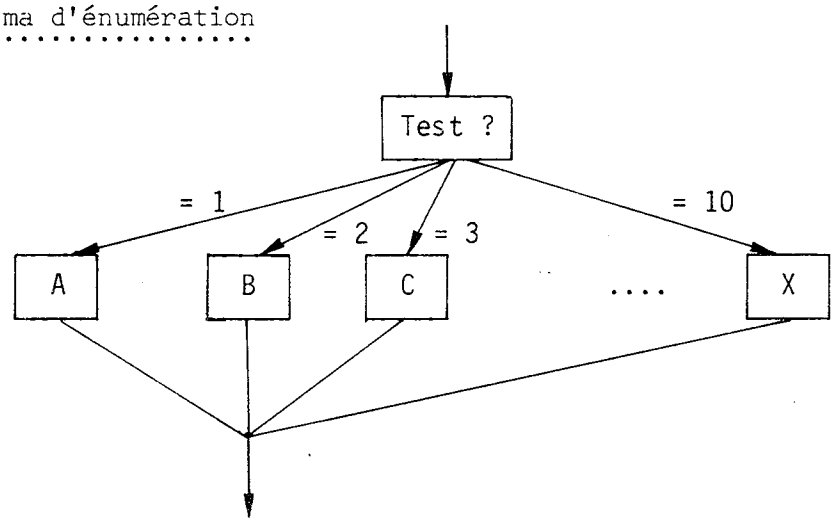
- L'évaluation de la condition permet de savoir si la liste de composants sera exécutée ou non

- En fin d'exécution de la liste de composants, il y a de nouveau évaluation de la condition. Ce schéma itératif a été introduit dans OASIS car le principe de la boucle est indispensable dans le cas d'applications informatiques, en particulier pour les circuits de redressement d'erreurs.

Le composant "suivant"

<suivant> → suivant <test><libellé> cas1 <liste de comp>
[... [cas2 <liste de comp>]
.....
[cas10 <liste de comp> ...]]
fsuiv

Schéma d'énumération



- L'énumération des cas1, cas2, ... est totalement ordonnée.
- L'évaluation du test permet de connaître le casi à activer
- Ce composant n'est pas indispensable mais il permet de garder une certaine clarté d'expression en évitant des imbrications de "si"
- La limite à 10 blocs possibles est uniquement fixée par souci de simplicité et d'efficacité à l'analyse syntaxique et à l'exécution

3.6.4. Les conditions

<condition> → <condition simple>|
<condition simple> et <condition>|
<condition simple> ou <condition>
<condition simple> →<cond.temps>|<décision>|<prog.base>

- L'exécution de toute condition simple a pour effet de mettre une variable SUCCES à la valeur 1 (oui) ou 2 (non)
- Un bloc de conditions peut être utilisé dans un composant "si" ou un composant "tantque"
- Il n'y a aucune parenthèse possible dans les expressions conditionnelles
- Il n'existe aucune priorité entre les opérateurs "et", "ou"
- L'évaluation d'une combinaison d'expressions conditionnelles se fait de la gauche vers la droite : ainsi l'expression C1 et C2 et C3 ou C4 et C5 ... sera évaluée avec les priorités exprimées par la notation parenthésée :

((((C1 et C2) et C3) ou C4) et C5)

- Il est bien évident que ces quelques règles limitent l'utilisation des expressions conditionnelles, mais elles ont été adoptées dans un souci de clarté pour l'utilisateur (l'enchaînement dynamique de tâches de gestion est soumis en général à des conditions simples) et dans un souci d'efficacité au niveau de la simulation
- Une condition simple est : - soit une condition temporelle
  - soit une décision
  - soit un programme d'analyse de l'état de la base.

La condition temporelle

<cond.temps> → t <numéro>

- C'est un programme Socrate écrit par l'analyste et rangé dans l'entité PROGRAMME avec un type "t" et un numéro. Ce programme contient uniquement des comparaisons de données stockées dans des caractéristiques temporelles du prototype fixées par l'utilisateur (des dates, des durées, etc ...)

- L'exécution de ce programme permet de mettre à jour le contenu d'une variable SUCCES a 1 ≡ OUI ou à 2 ≡ NON

- Ce programme est toujours exécuté automatiquement.

La décision

<décision> → d <numéro> / <groupe d'exécution>

- C'est une action d'un genre particulier dont le résultat est la réponse par oui ou non à une question. Elle est identifiée par un numéro

- Cette décision est prise par une personne appartenant au groupe d'exécution de l'action correspondante

- Selon la réponse à une action du type décision par oui ou par non, on valorisera la variable SUCCES à 1 ou 2.

Le programme de type base

<prog.base> → b <numéro> / <groupe d'exécution>
b <numéro> / auto

- C'est un programme Socrate écrit par l'analyste, rangé dans l'entité PROGRAMME avec un type "b" et un numéro

- Ce programme contient uniquement des tests sur le contenu de la base du prototype, et suivant l'état de cette base, permet de mettre la variable SUCCES à 1 ou à 2 : ce programme ne fait aucune mise à jour, ni édition, ni création de données dans la base. Il ne change pas l'état de la base. Il n'utilise pas les caractéristiques temporelles du prototype.

- Ce programme peut être activé soit automatiquement, soit par une personne appartenant à son groupe d'exécution.

3.6.5. Les tests

<test> → <n-décision>|<n-base>
<n-décision> → dn <numéro> / <groupe d'exécution>
<n-base> →bn <numéro> / <groupe d'exécution>|
bn <numéro> / auto

- Un test est un composant élémentaire utilisé dans le composant complexe "suivant"
- Un test est soit une décision à n-issues, soit un programme d'analyse de la base,qui renvoient un entier positif ou nul inférieur ou égal à n.
- Une n-décision est une action de type "dn" identifiée par son numéro
- Une n-base est un programme de type "bn" identifié par son numéro
- Si l'entier fourni par l'évaluation du test est égal à zéro aucun des cas1 ... casi ... casn qui suivent le test ne sera exécuté et on passera en séquence au composant qui suit le composant "suivant" dans lequel est incorporé le test
- Si l'entier fourni par l'évaluation du test est égal à "j", alors on exécutera le composant indiqué par le mot-clef "casj".

3.6.6. Remarque méthodologique

Pourquoi avoir séparé des conditions ou des tests qui dépendent

- du temps (t)
- d'une décision externe (d, dn)
- d'une évaluation de l'état de la base (b,bn) ?

Soulignons d'abord que cette distinction méthodologique de nature n'empêche pas évidemment de construire des conditions complexes par des opérateurs "et" et "ou" à partir de conditions simples.

On peut écrire si t10 et d15 ou b20 alors ....

pour représenter un choix qui dépende à la fois d'une condition temporelle d'une décision et de l'état de la base.

Toutefois, la contrainte syntaxique de définition de la nature des conditions simples et des tests oblige l'informaticien qui écrit le prototype à distinguer les paramètres d'évolution du système d'information qu'il simule à trois niveaux bien distincts.

- Les décisions humaines qui ne peuvent être, en aucun cas, automatisées et qui ont une influence sur le fonctionnement du système d'information.

- Les conditions purement temporelles qui caractérisent l'évolution du système d'information.

- Les conditions déduites des données stockées pour l'application.

L'exemple suivant permettra de juger l'importance de cette différentiation :

Dans le processus d'inscription d'un élève à l'Université, on peut dégager des conditions de différentes natures :

= décision humaine : celle de l'élève qui veut s'inscrire. Elle ne dépend que de sa propre initiative.

= condition temporelle : celle qui oblige l'élève à s'inscrire avant une certaine date.

= condition évaluée par rapport aux données de l'application informatique : l'obligation de présenter une inscription à des certificats compatible avec le cursus universitaire déjà obtenu, cursus enregistré dans le fichier historique de la scolarité.

3.6.7. La grammaire d'OASIS

R0 <définition de module> → DEFMOD <code module><libellé><Liste de composants> FINMOD

R1 <liste de composants> → <composant> [<composant>]*

R2 <composant> → <composant simple><libellé>|<composant complexe>

R3 <composant simple> → <ac><groupe> |<pg><affectation>|<md>

R4 <composant complexe> → indep <liste de composants> findep|
si <condition><libellé> alors <liste de composants>
[sinon <liste de composants>] fsi|
tantque <condition><libellé> faire<liste de comp> ftq|
suivant <test><libellé> cas1 <liste de composants> ...
[cas10 <liste de composants>] fsuiv

R5 <condition> → <condition simple>|
<condition simple> et <condition>|
<condition simple> ou <condition>

R6 <condition simple> → <tps>|<dec><groupe> |<base><affectation>|

R7 <test> → <ndec><groupe> |<nbase><affectation>|

R8 <affectation> → <groupe> | / auto

R9 <groupe> → / <code groupe>

R10 <tps> → t <numéro>

R11 <dec> → d <numéro>

R12 <base> → b <numéro>

R13 <ndec> → dn <numéro>

R14 <nbase> → bn <numéro>

R15 <pg> → p <code programme>

R16 <ac> → a <code action>

R17 <md> → m <code module>

R18 <libellé> → /* texte */

Remarque : Cette grammaire est quelque peu différente de celle présentée précédemment au niveau des règles R3, R8 et R9, mais ces différences ne changent évidemment rien aux spécifications du langage produit. C'est à elle que nous ferons référence dans les pages qui suivent.

3.7. EXEMPLES D'APPLICATION

Reprenons l'exemple du chapitre 2 (§§ 2.3.3. et 2.5.2.) sur la gestion d'un congrès pour décrire avec OASIS la structure de son prototype.

L'application décrite en OASIS

```
DEFMOD      M001      |* gestion des inscriptions à un congrès *|
        m   M002      |* définition du programme *|
        m   M003      |* appel aux communications *|
        m   M020      |* édition de la documentation *|
        a   A005 | SECRETARIAT |* envoi des bulletins d'inscription *|
        m   M004      |* procédure d'enregistrement tous les 15 jours
                         jusqu'à 2 jours du congrès avec édition des
                         listes provisoires *|
        m   M005      |* journée d'ouverture *|
        m   M101      |* édition des  listes définitives de participants
                         à la demande du responsable *|
        m   M009      |* bilan post-congrès, un mois après la fin du
                         congrès *|
FINMOD
```

Les modules M004, M005, M101, M009 dont l'exécution doit satisfaire des conditions (temporelles pour M004, M005, M009, 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 M001 : par exemple, au niveau du module M101, on aurait pu écrire :

```
si d 11 | GRESPONSABLE
   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   M020      |* édition de la documentation *|
      a   A001 | GRESPONSABLE |* décision sur le programme définitif *|
      indep
            a  A002 | GSTECHNIC |* édition de la plaquette programme *|
            m  M034   |* dessin et impression de la fiche d'ins-
                        cription *|
      findep
FINMOD
```

Les actions A003 (dessin) et A004 (impression) doivent être regroupées dans un module M034 de façon à pouvoir être globalement un élément dans un déroulement en parallèle avec l'action A002.

```
DEFMOD    M034       |* La fiche d'inscription *|
      a     A003 | GSTECHNIC |* dessin de la fiche d'inscription *|
      a     A004 | GSTECHNIC |* impression de la fiche d'inscription *|
FINMOD
```

```
DEFMOD    M004       |* enregistrement des participants *|
      tantque t 40 et t 41        |* tous les 15 jours jusqu'à 2 jours avant
                                     le congrès *|
         faire
            m M041     |* réception des inscriptions *|
            a A010 | SECRETARIAT |* envoi avis d'hébergement et de
                                     remboursement *|
            m M100     |* édition de listes provisoires *|
         ftq
FINMOD
```

```
DEFMOD    M041           |* réception des inscriptions *|
      a     A006   SECRETARIAT   |* réception postale des bulletins *|
      indep
            m M014   |* mise à jour du fichier des inscrits *|
            a A008 | SECRETARIAT |* réservation d'hôtel *|
      findep
FINMOD
```

```
DEFMOD    M014                |* mise à jour du fichier des inscrits *|
      a     A007 | PERF       |* perforation des bulletins reçus *|
      m     M140              |* mise à jour du fichier *|
FINMOD

DEFMOD    M140     |* mise à jour du fichier *|
      p   p003 | GSTECHNIC |* procédure de contrôle et de mise à jour *|
      p   p006 | auto      |* édition de la situation de paiement *|
      p   p005 | auto      |* édition demande de paiement du solde *|
      p   p004 | auto      |* messages d'erreurs *|
      tantque b 4 | auto    |* bilan des erreurs *|
          faire a A007 | PERF |* perforation *|
                p p003 | GSTECHNIC |* contrôle - mise à jour *|
                p p006 | auto      |* édition-1 *|
                p p005 | auto      |* édition-2 *|
                p p004 | auto      |* édition-3 *|
          ftq
FINMOD

DEFMOD    M100  |*  édition de listes provisoires *|
      p   p001 | auto |* édition de listes *|
FINMOD

DEFMOD    M101   |* édition des listes définitives à la demande de responsable *|
      si d 11 | GRESPONSABLE    |* décision du responsable *|
         alors  p  p001 | auto  |* édition de listes *|
      fsi
FINMOD
      ........
```

3.8. L'ANALYSEUR

L'objectif du langage OASIS étant de permettre une description des modules beaucoup plus rigoureuse que dans MACSI1, nous allons maintenant examiner successivement comment les spécifications écrites dans ce langage sont analysées syntaxiquement et quelle est la structure du code généré. Nous verrons ensuite l'utilisation faite de ce code pour conduire la simulation du prototype.

3.8.1. Principes de compilation

L'appel associatif

Notons tout d'abord que nous avons encore largement utilisé dans l'analyseur syntaxique DEFMOD la possibilité que nous offre le système Socrate-360 d'appeler associativement des programmes sélectionnés par des filtres sur leurs caractéristiques (cf. [37]).

Cet appel associatif s'exprime par une instruction :

"CALL Xi"

où Xi représente l'adresse présélectionnée d'un programme Socrate rangé dans l'entité SYSTEME (entité utilisée pour la gestion d'une base Socrate ...). Ainsi :

```
M X9 = UN PROGRAMME AYANT NOM = 'JOB' ; DE UN SYSTEME
CALL X9
```

est équivalent à l'instruction :

```
CALL JOB
```

On peut écrire aussi :

```
M X10 = UN PROGRAMME 7 DE UN SYSTEME
CALL X10
```

si JOB était le septième programme rangé dans l'entité programme.

Les classes syntaxiques

Actuellement, le langage OASIS est composé de 12 éléments de base qui sont : les actions (a), programmes (p), modules (m), conditions temporelles (t), décisions (d,dn), programmes-base (b,bn), indep, si, tantque, suivant.

Nous définissons 4 classes syntaxiques regroupant ces différents éléments :

* La première classe regroupe les 3 primitives de base définies par la règle R3 et associées aux mots-clefs "a", "p", "m".

* La deuxième classe est composée des 4 constructions élémentaires, définies par la règle R4, associées aux mots-clefs : "indep", "si", "tantque", "suivant".

* La troisième classe contient les 3 conditions simples définies dans la règle R6 : "t", "d", "b".

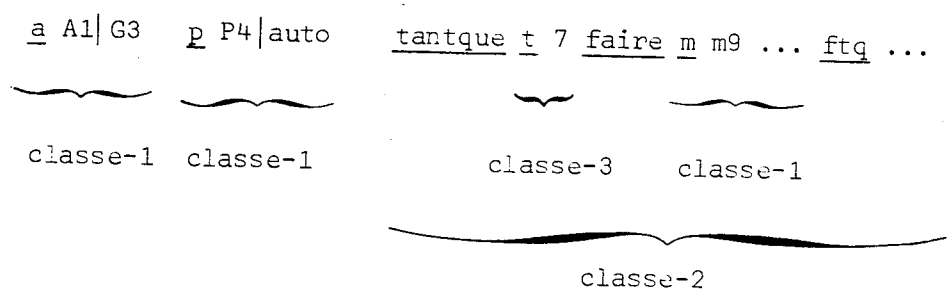
* Et la quatrième classe est réservée aux 2 tests "dn", "bn" définis en R7.

On peut dire qu'une phrase du langage OASIS (une définition d'un module) sera un ensemble de composants appartenant à la première ou à la deuxième classe ; les composants de la deuxième classe pourront contenir des éléments appartenant à la première, troisième ou quatrième classe.

Exemple :

Par souci de clarté, nous n'avons pas indiqué dans la grammaire et dans les exemples qui suivent l'emplacement des commentaires (|* ... *|) attachés à chaque composant d'une phrase OASIS.

DEFMOD M4



Les programmes d'analyse

Chaque élément de classe sera analysé par un programme particulier dont le nom sera le nom de la production (a, p, m, si, tantque, ...) et le niveau sera celui de la classe (l'indice de la classe). Tous ces renseignements peuvent se stocker aisément dans une entité Socrate :

```
Entité ROUTINE
début  TYPRT mot                |* nom du programme *|
   |   NIVRT de 0 à 100          |* niveau du programme *|
   |   PANA de 0 à 1000          |* numéro du programme d'analyse *|
   |   PINT de 0 à 1000          |* numéro du programme d'exécution *|
fin
```

Par exemple, on aura la routine de TYPRT = a, de NIVRT = 1, telle que PANA indique le numéro du programme Socrate, stocké dans l'entité PROGRAMME et qui réalise l'analyse d'une production "action".

a <code action> / <groupe d'exécution>

L'analyseur

La méthode d'analyse consiste alors à lire un mot de la phrase à analyser dans une variable ELT (élément courant) ; sachant que l'analyseur se trouve dans un certain état caractérisé par le niveau autorisé de classe syntaxique (1, 2, 3, 4 ...), on recherche le programme correspondant qui analysera toute la production. Si ce mot n'identifie aucune classe autorisée (aucune ROUTINE ayant typrt = elt et nivrt = X ; ...) alors il y a diagnostic d'erreur et passage au mot suivant.

Nous utilisons donc un principe où le mot lu nous permet, sans ambiguïté, de remonter à l'une des notions auxiliaires <action>, <programme>, ... de la grammaire.

L'analyse d'une liste de composants peut se faire de la façon suivante :

```
PROC ANALYSE
|   M ELT = EXT            |* lecture du premier mot de la liste *|
|   FAIRE
|   |   SI EXISTE UNE ROUTINE X9 AYANT TYPRT = ELT
|   |   |                          ET NIVRT = 1 OU NIVRT = 2 ;
|   |   |  |* Recherche de la routine d'analyse d'un composant dont le
|   |   |     mot-clef est ELT *|
|   |   |  ALORS  M Y10 = PANA de X9
|   |   |         M X10 = UN PROGRAMME Y10 de UN SYSTEME
|   |   |         CALL X10     |* analyse du composant correspondant *|
|   |   |         SI ARRET = 1 alors SORTIE   fin
|   |   |         ...
|   |   |  SINON  I  'ERREUR' |* ELT NE PEUT PAS ETRE UN DEBUT DE COMPOSANT
|   |   |                        DES CLASSES 1 OU 2 *|
|   |   |         ...
|   |   FIN
|   |   REFAIRE
|   FIN
|   ...
FPROC ?
```

Intérêt de ce principe d'analyse

Ce principe d'analyse offre le gros avantage de permettre facilement des modifications du langage OASIS.

* Si l'on veut modifier la syntaxe d'une primitive (une production), il suffit de modifier le programme d'analyse propre à cette primitive.

* Dans le cas de l'introduction d'une nouvelle primitive, en particulier si l'on souhaite construire d'autres composants complexes, il faut alors écrire un programme d'analyse de cette primitive, en définissant sa classe syntaxique afin de mettre à jour l'entité ROUTINE.

Principales règles d'analyse

Pour aider l'informaticien dans la conception et la mise au point de son prototype, nous avons fixé un certain nombre de règles afin que l'analyseur ne s'arrête pas à la première anomalie rencontrée dans la chaîne.

I) Dans la version actuelle de MACSI2, nous avons interdit les imbrications de règles. Ceci correspond à une modification de la règle R4 de la grammaire présentée, où la classe syntaxique <liste de composants> sera remplacée par <liste de composants simples>, en ajoutant une nouvelle règle :

R41, <liste de composants simples> → <composant simple>
[<composant simple>]*

Ainsi, un élément indep, alors, sinon, faire, cas i ne pourra être composé que d'actions (a), programmes (p) ou modules (m).

Par exemple, nous pourrons avoir le module M4 :

Définition 1 :

```
DEFMOD M4
      a  A1  |GDIRECTION
      suivant dn10|GR cas1 m M5
                     cas2 m M6
                     cas3 m M8
      fsuiv
FINMOD
```

Mais nous interdisons la définition suivante :

Définition 2 :

```
DEFMOD M4
       a  A1     |GDIRECTION|
       suivant dn10|GR cas1 m M5
                      cas2 indep m M7  a A4|G004 findep
                      cas3 m M8
       fsuiv
FINMOD
```

Cette définition devra être remplacée par la définition 1, complétée par la définition 3 suivante :

```
DEFMOD M6
       INDEP m M7 a A4|G004 FINDEP
FINMOD
```

Les imbrications de compositions ne pourront donc pas être implicites, mais seront explicites par déclaration de modules intermédiaires (M6 par exemple).

Cette restriction peut se justifier aisément au niveau méthodologique : un composant complexe (indep, si, tantque, suivant) représente toujours une unité fonctionnelle de traitement et,de façon naturelle, elle peut donc être représentée par un module.

II) Si au cours de l'analyse d'une primitive simple P1 (a, p, m, t, d, ...) l'analyseur rencontre le mot-clef d'une nouvelle primitive simple ou complexe P2 (a, p, m, ..., si, ...), alors la description de P1 est tronquée et déclarée incomplète ; l'analyse continue sur P2, si naturellement les niveaux (NIVRT) étaient compatibles avec l'état de l'analyseur.

Exemple :

```
a ENVOILETTRE6 | SECRETARIAT a XY  p P9 | auto ...
```

* l'analyse de l'action ENVOILETTRE6 se déroule normalement
* l'action Y est déclarée incomplète (sans affectation)
* l'analyse du programme P9 se fait correctement
...

III) Si dans un bloc condition (si ..., tantque ...) les opérateurs "et", "ou" sont omis, alors les conditions simples sont complètement analysées après avoir simplement indiqué cette ommission d'opérateur et l'analyse continue.

Exemple :

si t 4 b 9 ou d 10 | G15 alors ...

* t 4 analysé correctement
* b 9 tronqué : pas d'affectation
* omission de l'opérateur entre t 4 - b 9
* analyse correcte de d 10
...

IV) Au cours de l'analyse d'un composant complexe P1 (indep, si, tantque, suivant), les mots-clefs de composition (et, ou, alors, faire, sinon, fsi, ...) sont prioritaires par rapport aux éléments constituants des composants simples (a, p, m, t, d, dn, bn, b).

Exemple :

si t 4 et d 9 alors ...

* la décision d 9 est déclarée incomplète (erreur d'affectation).
* cependant, "alors" sera pris en compte et ferme le bloc condition.

3.8.2. Structure associée au pseudo-code

Entité MOD

Elle permet de stocker le code résultat de l'analyse syntaxique des chaînes de descriptions modulaires des MODULES.

Elle est définie par :

```
Entité MOD
début MDVALID de 0 à 100
    |   MDTYP de 0 à 100
    |   MDCD mot discr
    |   MDNUM de 0 à 10000
    |   MDSOURCE texte 20
    |   MDNBCP de 0 à 20
    |   Entité COMPOSANT
    |   début CPNUM de 0 à 20
    |       |   CPSUIV référence un COMPOSANT de un MOD
    |       |   CPCHAIN de 0 à 100
    |       |   CPTYP de 0 à 100
    |       |   CPEXEC de 0 à 10000
    |       |   CPCOD mot
    |       |   CPCODNUM de 0 à 10000
    |       |   CPGRP mot
    |       |   CPCDTS référence un MOD
    |       |   CPSUIV1 référence un MOD
    |       |   CPSUIV2 référence un MOD
    |       |   ......
    |       |   CPSUIV10 référence un MOD
    |   fin
fin
```

Un MOD est défini par son nom (MDCD) s'il est associé à un MODULE défini par l'utilisateur, sinon par son numéro (MDNUM s'il a été introduit par le compilateur pour regrouper des blocs homogènes d'informations (une expression conditionnelle, un bloc sinon-fsi, ...).

Le type du MOD est rangé dans MDTYP.

MDTYP = 1, module défini par l'utilisateur avec les instructions "m" et "defmod"

= 2, bloc de règles :

indep <...> findep
alors <...> sinon, sinon <...> fsi
faire <...> ftq
cas1 <...> cas2, cas2 <...> cas3, ...

= 3, bloc de conditions : si <...> alors
tantque <...> faire

= 4, test : suivant <...> cas1

Dans MDSOURCE, on retrouve le texte en OASIS de la description du module concerné.

Un MOD est défini par ses composants (COMPOSANT) dont le nombre est rangé dans MDNBCP.

Un COMPOSANT est défini par :

* son type (CDTYP), 1 = action, 2 = programme, 3 = module, 11 = indep, 12 = si, 13 = tantque, 14 = suivant, 20 = temps, 21 = décision, 22 = prog. base, 30 = n-décision, 31 = n-prog-base.

* ses paramètres : son code alphanumérique (CPCOD) ou numérique (CPCODNUM), son groupe d'exécution éventuel (CPGRP), son programme d'exécution associé (CPEXEC).

* son numéro d'ordre (CPNUM) dans le MOD.

Un composant peut être suivi d'un autre composant défini par CPSUIV et par CHCHAIN qui précise la nature du chaînage avec le composant suivant : 0 = aucun suivant, 1 = et, 2 = ou, 90 = indep, 99 = séquentiel.

Lorsqu'un composant est complexe (indep, si, tantque, suivant), des pointeurs permettent de le chaîner avec les blocs de règles ou de conditions.

CPCDTS : test d'un "suivant", ou conditions d'un "si" ou "tantque"
CPSUIV1 : bloc, indep, faire, alors ou cas1
CPSUIV2 : bloc sinon ou cas2
CPSUIV3 : bloc cas3
...
CPSUIV10 : bloc cas10.

Génération de pseudo-code

Lorsqu'un utilisateur définit un module par la commande DEFMOD, il donne le nom de ce module, un libellé explicatif, et une description modulaire (une liste de composants). Après vérifications d'unicité du code de ce module, le système MACSI2 :

- enregistre ce nouveau module, son nom, son libellé, sa liste de composants
- appelle le système OASIS :
    * il analyse la liste de composants en générant le pseudo-code correspondant
    * après avoir vérifié que tous les composants sont complètement définis et après avoir appliqué des règles de cohérence, l'analyse est déclarée "bonne" ; alors le pseudo-code de ce module devient valide pour une éventuelle interprétation.

C'est le programme DEFMOD, dans le système OASIS, qui gère la définition d'un module ; il appelle l'analyseur MODANA qui est chargé de la vérification syntaxique de la liste des composants du module ; si MODANA ne rencontre aucune anomalie syntaxique, alors DEFMOD appelle le programme BILMOD, qui évalue la validité sémantique du pseudo-code généré par MODANA et définira si ce pseudo-code est interprétable. Si l'analyseur MODANA retourne des listes d'erreurs, alors le pseudo-code est détruit par le programme TUMOD.

Le programme MODANA a 3 fonctions principales :

- analyser syntaxiquement une liste de composants
- générer le pseudo-code correspondant
- retourner éventuellement un bilan d'erreurs.

I) L'analyse se fait selon les principes définis dans les pages précédentes : l'analyseur se trouvant dans un certain état, la lecture d'un mot définit sans ambiguïté le nouvel état de l'analyseur.

Exemple : Soit la chaîne à analyser :

a A7 | G07 p P6 a a9 ...

partie déjà analysée

- L'analyseur se trouve dans l'état qui a été défini par l'analyse d'un composant appartenant à la première classe : le composant p ...
- La lecture du mot suivant : "a"
  - indique à l'analyseur qu'il rencontre un composant de même niveau appartenant à la même première classe
  - déclenche un diagnostic d'erreurs : composant précédent incomplet ...
  - fait changer d'état l'analyseur qui passe maintenant à l'analyse d'une production a ....

II) La génération de pseudo-code se fait parallèlement à l'analyse et il est stocké dans la structure de l'entité MOD, selon des règles que nous allons illustrer à partir des modules M020 et M004 de l'exemple.

```
DEFMOD  M020
        a A001 | GRESPONSABLE
        indep a A002 | GSTECHNIC  m  M034  findep
FINMOD
DEFMOD  M004
        tantque t 40 et t 41
        faire m M041 a A010 | SECRETARIAT m M100 ftq
FINMOD
```

a - Pseudo-code généré à partir de l'exemple

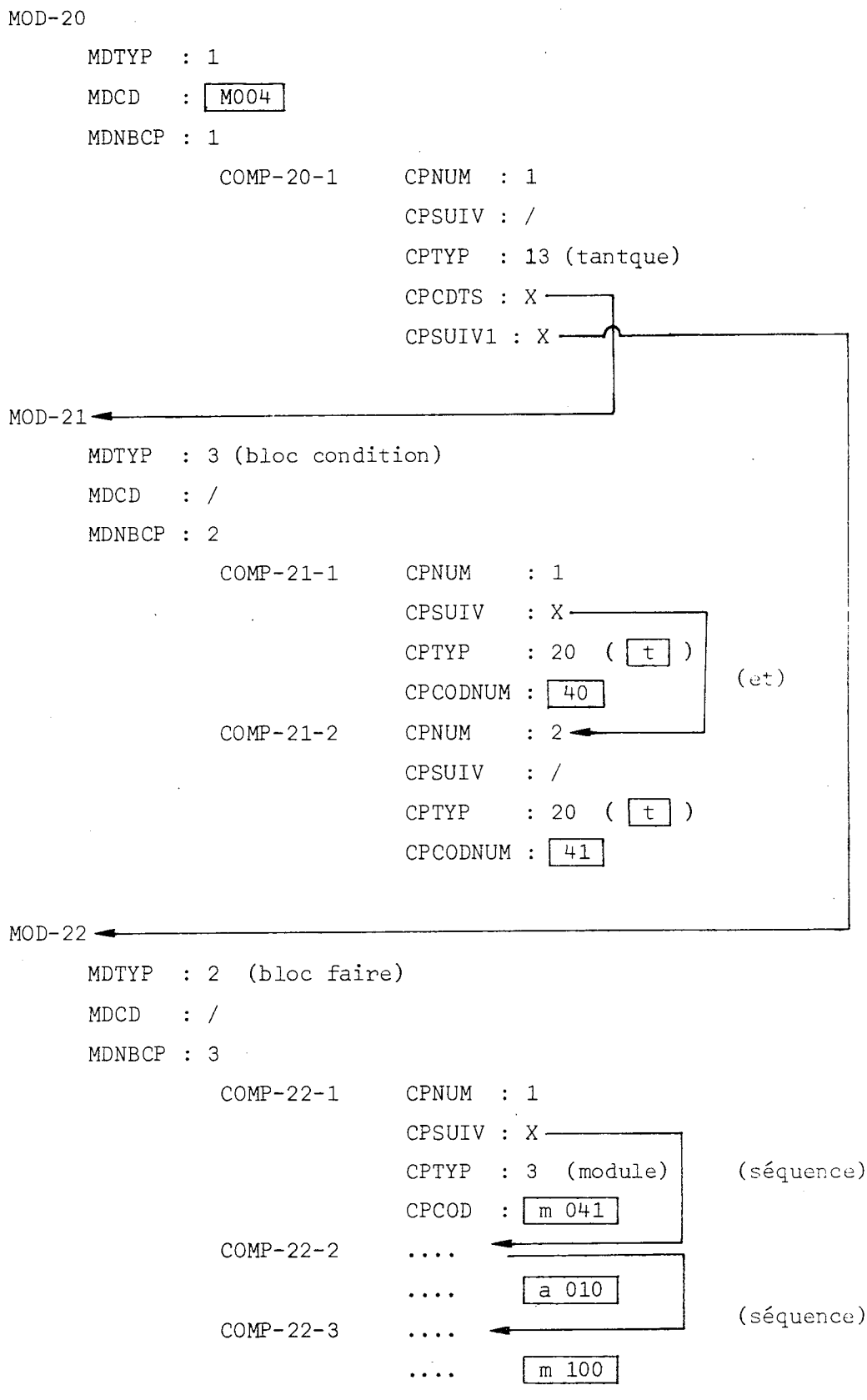
```
MOD-10

        MDTYP   : 1  (module)
        MDCD    : [M020]
        MDNBCP  : 2

                COMP-10-1   CPNUM    : 1
                            CPSUIV   : X ──────────┐
                            CPTYP    : 1  (action) │
                            CPCOD    : [a001]      │   (séquence)
                            CPGRP    : GRESPONSABLE│
                COMP-10-2   CPNUM    : 2 ◄─────────┘
                            CPSUIV   : /
                            CPTYP    : 11  (indep)
                            CPSUIV1  : X ──────────┐
                                                   │
MOD-11 ◄───────────────────────────────────────────┘

        MDTYP   : 2  (bloc)
        MDCD    : /
        MDNBCP  : 2

                COMP-11-1   CPNUM    : 1
                            CPSUIV   : X ──────────┐
                            CPTYP    : 1  (action) │
                            CPCOD    : [a002]      │   (indep)
                            CPGRP    : Gstechnic   │
                COMP-11-2   CPNUM    : 2 ◄─────────┘
                            CPSUIV   : /
                            CPTYP    : 3  (module)
                            CPCOD    : [m034]
```


MOD-20
MDTYP : 1
MDCD : M004
MDNBCP : 1
COMP-20-1
CPNUM : 1
CPSUIV : /
CPTYP : 13 (tantque)
CPCDTS : X
CPSUIV1 : X
MOD-21
MDTYP : 3 (bloc condition)
MDCD : /
MDNBCP : 2
COMP-21-1
CPNUM : 1
CPSUIV : X
CPTYP : 20 (t)
CPCODNUM : 40
(et)
COMP-21-2
CPNUM : 2
CPSUIV : /
CPTYP : 20 (t)
CPCODNUM : 41
MOD-22
MDTYP : 2 (bloc faire)
MDCD : /
MDNBCP : 3
COMP-22-1
CPNUM : 1
CPSUIV : X
CPTYP : 3 (module)
CPCOD : m 041
(séquence)
COMP-22-2
....
....
a 010
COMP-22-3
....
....
m 100
(séquence)

b - Commentaires

- Il y a création d'une réalisation de l'entité MOD pour enregistrer le module M020 et sa description, réalisation que nous appellerons MOD-10.

- Le module M020 est composé de 2 composants :

. a a001 | Gresponsable

et . indep a a002 | GStechnic m m034 findep

- La réalisation MOD-10 sera complétée successivement en cours d'analyse, de 2 réalisations de la sous-entité COMPOSANT, nous les appellerons COMP-10-1 et COMP-10-2.

- L'analyse du composant [a a001 | Gresponsable] permet de générer COMP-10-1 et de mettre à jour son contenu en enregistrant que :

c'est le premier composant de MOD-10	CPNUM = 1
c'est une action	CPTYP = 1
de nom a001	CPCOD = a001
son exécution sera assurée par	CPGRP = GRESPONSABLE
et gérée par le programme système	CPEXEC = ...

- L'analyse du deuxième composant [indep a a002 ... m m034 findep] génère COMP-10-2, chaîne COMP-10-1 et COMP-10-2 par les caractéristiques CPSUIV et CPCHAIN de COMP-10-1, et enfin met à jour COMP-10-2 :

CPNUM = 2	2ème composant de MOD-10
CPTYP = 11	c'est un indep ...
CPEXEC = ...	

Il y a maintenant création d'une nouvelle réalisation de MOD : MOD-11 qui permettra de stocker les différents éléments de la simultanéité ; et chaînage de COMP-10-2 avec MOD-11 par CPSUIV1 de COMP-10-2.

- L'analyse de "indep" continue et va générer successivement 2 composants COMP-11-1 et COMP-11-2 pour stocker [a a002 | GSTECHNIC] et [m M034] ...

- L'analyse du module M004 générera une réalisation MOD-20 constituée d'un seul composant COMP-20-1 (tantque ... ftq).

- COMP-20-1 sera mis à jour par "l'analyseur du tantque"

CPNUM = 1 1er composant

CPTYP = 13 tantque

Il y aura création de 2 réalisations de MOD : MOD-21 et MOD-22 pour stocker successivement le bloc condition (t 40 et t 41) et le bloc faire ... ftq.

MOD-21 sera chaîné à COMP-20-1 par CPCDTS
et MOD-22 par CPSUIV1.

c - *Règles principales d'analyse*

- L'analyse d'un composant simple a, p, m, t, d, b, dn, bn est suivi de la création et mise à jour d'une réalisation de COMPOSANT.

- L'analyse d'un composant complexe (indep, si, tantque, suivant) est suivie de la création et mise à jour d'une réalisation de COMPOSANT et d'autant de réalisations de MOD (et de COMPOSANTS correspondants) que de blocs homogènes d'informations contenus dans ce composant complexe.

* un "indep" donnera naissance à un "MOD" regroupant les différents COMPOSANTS s'exécutant collatéralement.
* un "tantque" : 2 réalisations de "MOD", l'une pour regrouper les conditions, l'autre pour le bloc faire ... ftq.
* un "si" : 2 ou 3 réalisations de "MOD" ; le bloc des conditions, le bloc "alors" et le bloc "sinon" éventuellement.
* un "suivant" : n+1 réalisations de "MOD" pour "n" cas.

III) Le programme BILMOD est activé, après l'analyse,d'une chaîne ne contenant aucune erreur syntaxique, et il est chargé de faire des évaluations :

* Tout composant de type "m" a-t-il été défini dans l'entité MOD (définition et description) ?

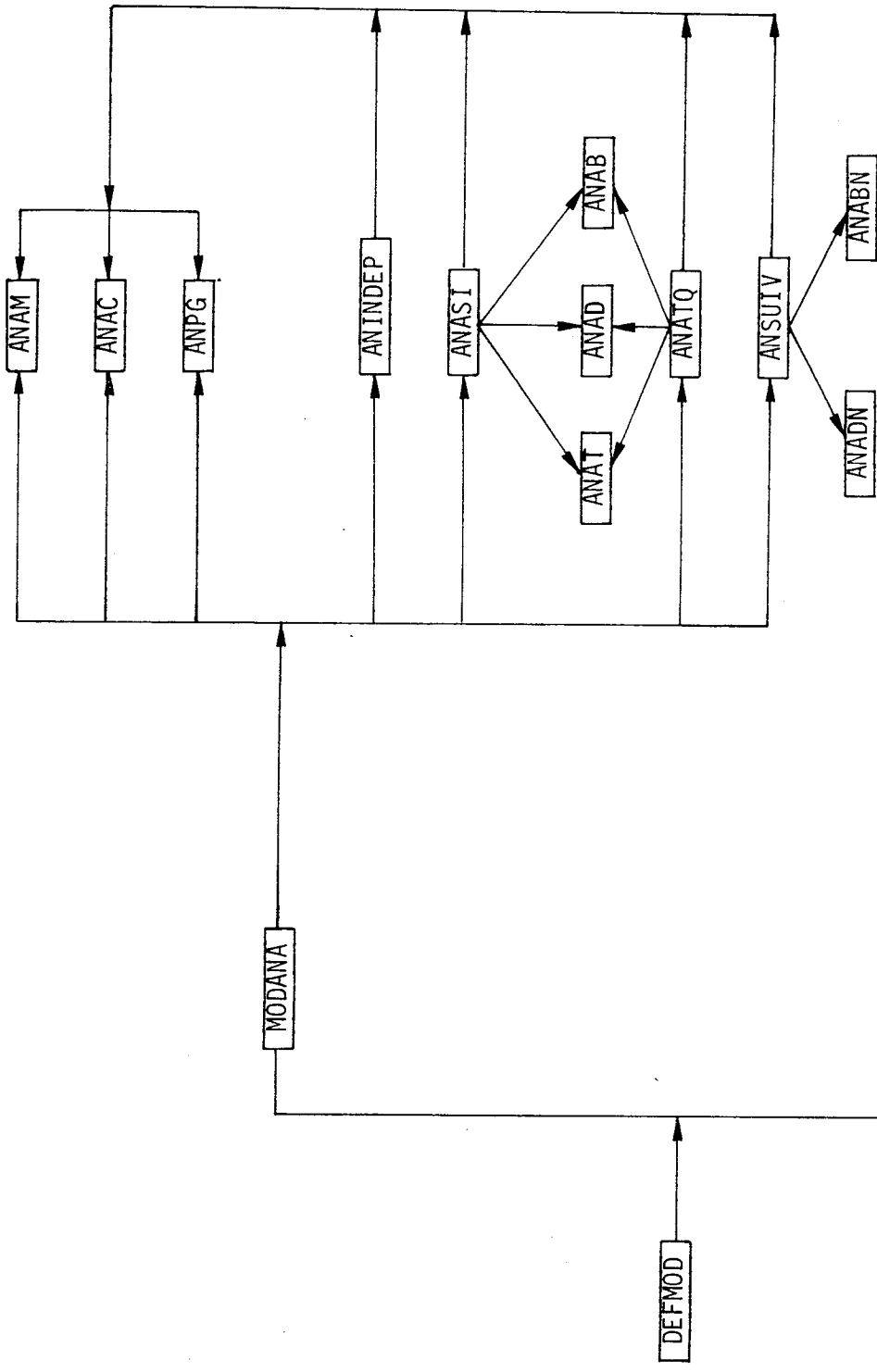
* Tout composant de type "a", "d", "dn", "p", "b", "bn", "t" a-t-il été défini dans une entité ACTION ou PROGRAMME et a-t-il une affectation (groupe d'exécution) ?

...

Le programme BILMOD édite ainsi un certain nombre de listes de diagnostics concernant :

- les tâches non affectées
- les tâches mal affectées
- les éléments cités non définis
- les éléments incomplets.

Si ces listes sont vides, BILMOD déclare le module valide pour l'exécution (MDVALID = 99).



SCHEMA D'APPEL DES PROGRAMMES D'ANALYSE

Rôle des différents programmes d'analyse

ANAM : analyse de m <code module>
ANAC : analyse de a <code action> / <groupe d'exécution>
ANPG : analyse de p <code programme> / <groupe d'exécution ou auto>
ANAT : analyse de t <numéro de programme>
ANAD : analyse de d <numéro d'action> / <groupe d'exécution>
ANADN : analyse de dn <numéro d'action> / <groupe d'exécution>
ANAB : analyse de b <numéro de programme> / <groupe d'exécution ou auto>
ANABN : analyse de bn <numéro de programme> / <groupe d'exécution ou auto>
ANINDEP : indep <liste de composants simples> findep
ANATQ : tantque <condition> faire <liste de composants simples> ftq
ANASI : si <condition> alors <liste comp> [sinon <liste de comp>] fsi
ANSUIV : suivant <test> cas1 ... cas2 ... [cas10 ...] fsuiv

3.9. LA SIMULATION

3.9.1. Principes généraux

Rappelons que l'objectif essentiel retenu dans MACSI2 est d'évaluer, après une exécution du prototype considéré comme simulation du projet, le degré de satisfaction de ses utilisateurs afin de faire les modifications éventuellement souhaitables. Les objectifs de cette simulation sont entre autres :

- familiariser les utilisateurs avec l'application
- tester les enchaînements spécifiés
- mesurer la bonne compréhension des actions définies
- faire prendre conscience de l'organisation du travail et des responsabilités de chaque groupe d'exécution.

C'est par cette simulation que le prototype prend toute sa valeur. Pour la conduire, il nous faut un système d'exécution qui permette un déroulement pas à pas de l'application, en exécutant les tâches élémentaires, en provoquant une interruption après chaque exécution élémentaire (action, décision, programme, ...), en donnant des informations nécessaires à l'utilisateur (trace de l'exécution, ...).

Nous avons envisagé 2 systèmes de simulation : un système-maître et un système-esclave ; mais avant de décrire ces systèmes, définissons les principes d'exécution des différents composants en rappelant que le "pseudo-code" généré par l'analyseur est directement interprétable : pour chaque COMPOSANT analysé, le pseudo-code contient donc le numéro du programme d'interprétation correspondant (CPEXEC de un COMPOSANT de un MOD).

Rôle du groupe d'exécution

Dans la description d'un module en langage OASIS, à chaque composant de type a, d, dn, p, b, bn est associé un code groupe identifiant un groupe d'exécution. Un groupe d'exécution est un ensemble de personnes assurant toutes les fonctions du groupe. Une personne peut appartenir à un ou plusieurs groupes d'exécution.

L'activation d'un composant associé à un groupe d'exécution ne pourra se faire que sous le contrôle d'une personne (vérification de son mot de passe) appartenant à ce groupe.

Exécution d'une action (a) de code CDAC

Cette exécution ne peut être lancée que par une personne appartenant à son groupe d'exécution. Elle est réalisée par l'appel d'un programme standard demandant à la personne "en ligne" si l'action "CDAC" a été réalisée. La personne peut répondre "oui" (alors l'action est considérée comme exécutée), ou répondre "quoi", alors le programme lui donnera des précisions sur la nature de cette action CDAC et réitèrera sa question jusqu'à l'obtention d'une réponse "oui", sinon la simulation sera bloquée au niveau de cette action.

Prise d'une décision (d, dn)

Le principe est le même que pour l'exécution d'une action (a), mais ici la personne appartenant au groupe d'exécution concerné devra répondre uniquement par "oui" ou "non" suivant la décision prise (d) ou par "0", "1", "2", ..., "n" pour une n-décision (dn). L'exécution d'une décision est considérée comme terminée à la première réponse valide obtenue.

Exécution d'un programme (p)

Cette exécution peut être lancée soit après accord d'une personne du groupe d'exécution auquel il a été rattaché, soit automatiquement sous le contrôle du système qui supervise la simulation.

Exécution d'un programme (t, b, bn)

C'est le même principe que pour l'exécution de programme de type p (sauf les programmes "t" qui sont toujours déclenchés automatiquement). Mais l'exécution de ces programmes devra retourner une valeur à la variable SUCCES (1 = oui ou 2 = non pour t et b, 0, 1, 2,., n pour bn).

Exécution d'un module (m)

C'est l'appel automatique de ce module, suivi par l'exécution du composant d'entrée de ce module. L'exécution d'un module est terminée lorsque tous ses éléments de sortie ont été exécutés.

Exécution d'un "suivant"

Il y a d'abord exécution du test (dn ou bn) conditionnant l'aiguillage puis exécution d'un bloc "cas i".

Exécution d'un "indep"

Dans la version actuelle, c'est l'exécution séquentielle de tous les composants dans l'ordre où ils sont décrits ; mais il est possible (toujours en système monoprocesseur) d'envisager une exécution "imbriquée" des composants : par exemple, soit la structure suivante :

```
* indep m m4 m m5 m m6 findep
* m4 ≡ a a40, p p4
* m5 ≡ a a5
* m6 ≡ a a7, m m70, p p6
* m70 ≡ a a10, a a11
```

Elle pourrait donner la trace :

a7, a5, a40, a10, a11, p4, p6

ou :

a40, a5, a7, a10, a11, p6, p4, etc ....

Exécution d'un "tantque"

C'est l'exécution des conditions, suivie éventuellement (selon la valeur conditionnelle) de l'exécution du bloc "faire" (exécution qui sera suivie de l'exécution du "tantque").

Exécution d'un bloc de conditions

C'est l'exécution de chaque condition simple de la gauche vers la droite avec évaluation de l'expression conditionnelle.

Simulation en mode "prototype-maître"

Dans ce mode, le système MACSI2 dirige alors la simulation comme suit :

* Une personne donne son mot de passe et demande la simulation numéro X d'une application Y.

* Si cette simulation X(Y) existe, nous sommes dans le cas d'une reprise d'une exécution déjà commencée et qui a été interrompue ; sinon il faut ouvrir cette simulation, si l'application Y est valide pour l'exécution.

* L'ouverture d'une simulation est la recherche de la première tâche élémentaire exécutable (l'entrée de l'application).

* La reprise d'une simulation recherche dans son "contexte d'exécution" le point d'interruption et définit alors la tâche suivante à exécuter.

* A la fin de l'exécution d'une tâche élémentaire, l'utilisateur a la possibilité de continuer, d'annuler ou de suspendre la simulation en cours, ou de demander une trace des exécutions faites.

* L'exécution de la dernière tâche de l'application est suivie de la fermeture de la simulation.

* L'exécution d'une tâche élémentaire a, p, t, d, dn, b, bn se fait en 2 phases :

Phase 1 : L'utilisateur tape son mot de passe dont le système vérifie la validité et la cohérence avec le code du groupe d'exécution affecté à cette tâche (sauf dans les cas de déclenchement automatique ou de conditions temporelles t).
Si ce mot de passe est valide, alors le système passe à la phase 2, sinon il génère une interruption, envoie un diagnostic suivi d'une suspension de la simulation.

Phase 2 : Le système lance l'exécution du programme associé à la tâche.
Suivant les tâches a, d, dn, p, b, bn ou t, ce sera un programme standard système ou un programme spécifique du prototype.

Simulation en mode "prototype-esclave"

Selon ce mode, le système MACSI2 contrôle les exécutions demandées par les utilisateurs.

a) Une personne donne son mot de passe et demande l'exécution d'une tâche élémentaire TE d'une application AP.

b) Le système recherche la simulation en cours pour cette application AP

- il vérifie que la tâche TE est une tâche exécutable à cet instant (elle suit le dernier point d'interruption)
- il vérifie la validité du mot de passe par rapport au groupe d'exécution affecté à TE
- alors il lance l'exécution de TE
  (ce qui équivaut à une reprise de simulation en mode maître).

c) Si le système ne trouve pas de simulation en cours pour l'application AP, et si cette application est valide pour la simulation, alors il vérifie que TE est la première tâche exécutable de l'application et il vérifie la validité du mot de passe puis exécute TE (équivalent à une ouverture de simulation).

d) Si les cas b et ou c sont impossibles, alors le système en indique les raisons :
- application invalide
- TE non exécutable en ce moment
- mot de passe incohérent avec TE
- ....

e) Après chaque exécution de tâche élémentaire, le système génère automatiquement une suspension de la simulation.

Quel que soit le mode de simulation, la simulation fait toujours appel aux sept fonctions de base : ouverture, reprise, suspension, continuation, annulation, fermeture, trace.

Déroulement d'une simulation en mode maître

Soit l'application EXEMP décrite par le module M6 composé de

```
defmod M6  a  A1 | G1
           si  t 9  et  b 4 | G3   alors  a  A9 | G5
                                  sinon  m  m13  m  m40  fsi
           p  p15 | auto  ...
```

Le système de simulation en mode maître, SIMAP, est appelé à partir du système MACSI2.

```
→ simap
  PASSWORD ?
→ dupont
  NOM de L'APPLICATION ?
→ exemp
  NUMERO de la SIMULATION ?
```

```
→ 1
  OUVERTURE de la SIMULATION : 1 de l'APPLICATION : EXEMP
  DECRITE par le MODULE : M6
  M6 = A A1
       SI T 9 ET B4 | G3 ALORS A A9
                    SINON M M13 M M40
       FSI
       P P15 ...
  FONCTION (CONTINUER, SUSPENDRE, ANNULER, TRACE) ?
→ continuer
  PASSWORD ?
→ dupont                                (vérification que dupont appartient à G1)
  AVEZ-VOUS EXECUTE L'ACTION : A1 ?
→ quoi ?
  L'ACTION A1 C'EST : .......
  AVEZ-VOUS EXECUTE L'ACTION : A1 ?
→ oui
  FONCTION (CONTINUER, SUSPENDRE, ANNULER, TRACE) ?
→ continuer
           (exécution automatique de t 9)
  FONCTION (CONTINUER, SUSPENDRE, ANNULER, TRACE) ?
→ continuer
  PASSWORD ?
→ dupont
           (vérification que dupont appartient à G3, exécution de b 4,
            évaluation de t 9 et b 4 : vrai par exemple)
  FONCTION (CONTINUER, SUSPENDRE, ANNULER, TRACE) ?
→ continuer
  PASSWORD ?
→ durand              (durand appartient à G5)
  AVEZ-VOUS EXECUTE L'ACTION : A9 ?
→ oui
  FONCTION (CONTINUER, SUSPENDRE, ANNULER, TRACE) ?
```

```
→ trace
  TRACE DE LA SIMULATION : 1 de l'APPLICATION : EXEMP
  ACTION A1 : DUPONT
  PROG T 9 : AUTO
  PROG B 4 : DUPONT
  ACTION A9 : DURAND
  FONCTION (CONTINUER, SUSPENDRE, ANNULER, TRACE) ?
→ suspendre
  SUSPENSION DE LA SIMULATION : 1 de l'APPLICATION : EXEMP
  (retour au système MACSI2).
```

3.9.2. Structure des données nécessaires à la simulation

Entité CONTEXTE

1. Une réalisation de cette entité permet d'exécuter un ensemble de traitements qui sont nécessairement regroupés par un "MOD" au niveau du pseudo-code. Son existence est liée à la durée d'exécution de ce "MOD".

Elle est définie par :

```
Entité CONTEXTE
début  CTNUM de 0 à 1000
  |    CTYP de 0 à 100
  |    CTMOD référence un MOD
  |    CTCOMP référence un COMPOSANT de un MOD
  |    CTCPSUIV référence un COMPOSANT de un MOD
  |    CTFILS référence un CONTEXTE
  |    CTPERE référence un CONTEXTE
  |    CTCOMPERE référence un COMPOSANT de un MOD
  |    CTEVAL de 0 à 100
  |    CTCHAIN de 0 à 100
fin
```

2. Cette structure a été choisie de façon à permettre aux modules (MOD), d'être réentrants et interruptibles ; un CONTEXTE contient totalement l'état d'avancement de l'exécution d'un MOD :

- MOD en cours : CTMOD
- dernier composant exécuté : CTCOMP
- composant suivant à exécuter : CTCPSUIV.

3. Un contexte, lorsqu'il n'est pas associé au MOD courant de l'application (CTYP = 10) est chaîné avec le contexte appelant (CTPERE) en indiquant le composant d'appel (CTCOMPERE). Le chaînage inverse étant réalisé par CTFILS.

4. Un CONTEXTE étant toujours associé à un "MOD", on peut lui affecter un type (CTYP).

CTYP = 10 ≡ "mod" application, contexte principal
11 ≡ "mod" m, module appelé
21 ≡ indep
22 = tantque ; 23 = si ; 24 = suivant
30 = alors, sinon ; 31 = faire ; 32 = cas i

5. Dans le cas du contexte de type "si" ou "tantque" (CTYP = 23 ou 22), l'analyse et l'exécution du "mod" regroupant les conditions et opérateurs (et, ou) utilisent les caractéristiques suivantes :

- CTEVAL qui contient la valeur booléenne de la chaîne des conditions déjà exécutées (CTEVAL = 1 : oui, = 2 : non)

- CTCHAIN indiquant le type du chaînage avec la condition suivante (CTCHAIN = 0 : fin du bloc condition, = 1 : et, = 2 : ou). L'évaluation d'une chaîne de conditions se faisant de la gauche vers la droite sans parenthèses comme pour la priorité entre les opérateurs booléens "et", "ou".

Entité APPLICATION

1. Elle permet de stocker les caractéristiques des différents prototypes spécifiés dans MACSI2. Elle est définie par :

```
Entité APPLICATION
début  APCD mot
  |    APDEF texte
  |    APDESCR mot
  |    APVALID de 0 à 100
  |    ....
fin
```

2. Une application est définie par un code (APCD), par un texte explicatif (APDEF).

La caractéristique APDESCR indique le code du module associé à cette application. Pour lancer en exécution une application, cette dernière doit être entièrement définie (APVALID = 99).

Entité SIMULATION

1. Elle permet de stocker les simulations réalisées **ou** en cours de réalisation des différentes applications.

Elle est définie par :

```
Entité SIMULATION
début  SIMDEB de 750101 à 991231
  |    SIMFIN de 750101 à 991231
  |    SIMQUOI référence une APPLICATION
  |    SIMNUM de 0 à 1000
  |    SIMCTX référence un CONTEXTE
  |    SIMESPION référence une TRACE
fin
```

2. Une simulation est associée à une application (SIMQUOI) et possède un numéro (SIMNUM). Pour une même application, il est possible d'avoir ainsi plusieurs simulations.

3. Les dates de début (SIMDEB) et de fin ou de suspension (SIMFIN) de simulation sont enregistrées.

4. Une simulation en cours est toujours associée à un contexte principal (SIMCTX) tel que CTYP = 10 et pointant sur le "MOD" décrivant l'application concernée.

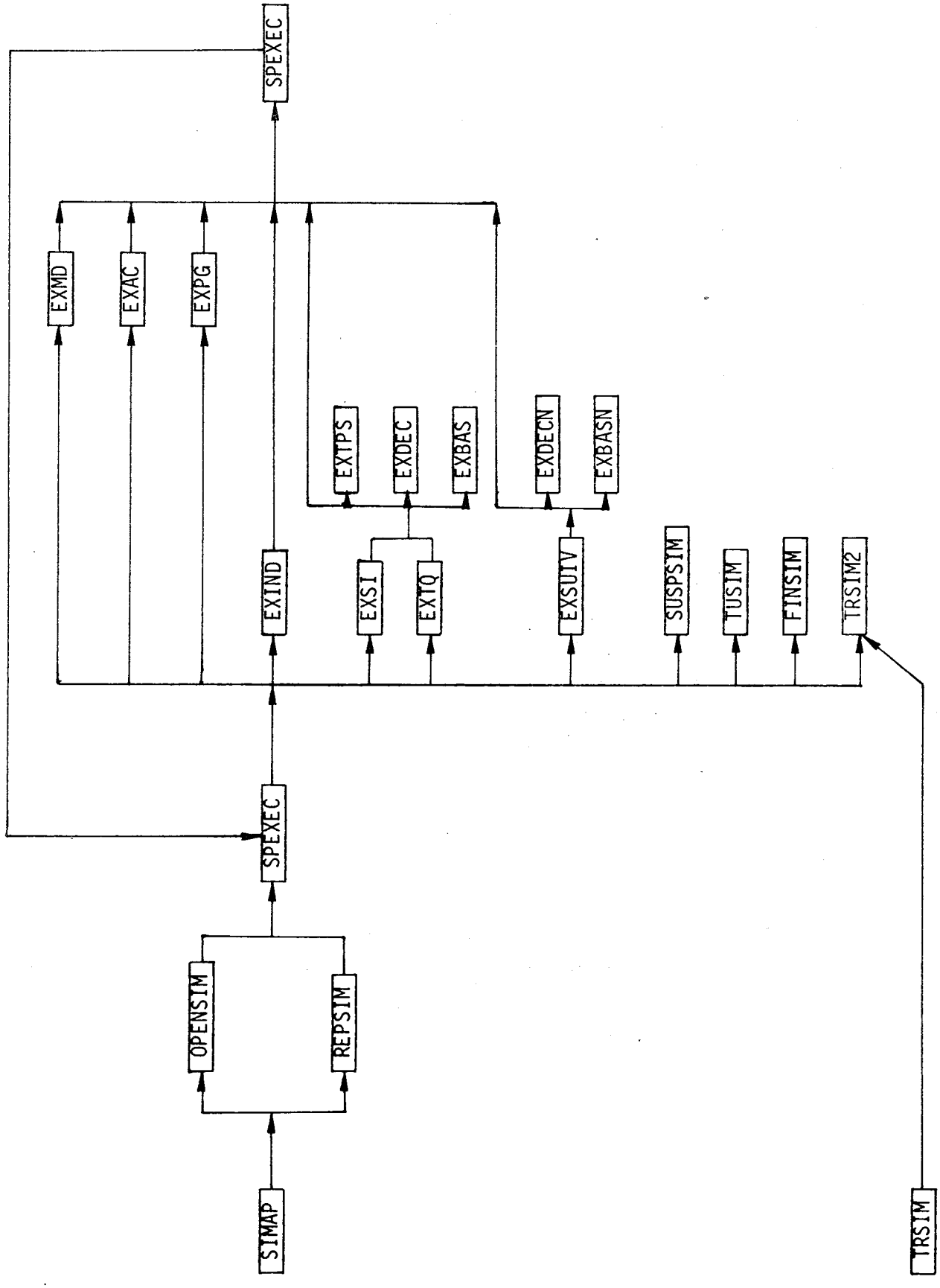
5. Une trace (SIMESPION) de la simulation est conservée.

Entité TRACE

Elle est définie par :

```
Entité TRACE
début  TRQUOI référence une SIMULATION
  |        TRNUM de 1 à 10000
  |        entité CHEMIN
  |        début  CHTACHE mot
  |          |        CHTYP de 0 à 100
  |          |        CHPERS mot
  |        fin
fin
```

Elle permet de stocker pour chaque simulation (TRQUOI) la liste (CHEMIN) des composants élémentaires exécutés en indiquant leur nom (CHTACHE), leur nature (CHTYP = 1 : a, = 2 : p, = 20 : t, = 21 : d, = 22 : b, = 30 : dn, = 31 : bn) et l'exécutant (CHPERS).



SCHEMA D'APPEL DES PROGRAMMES D'EXECUTION

Rôle des différents programmes de simulation

SIMAP : initialisation d'une demande de simulation
OPENSIM : ouverture d'une nouvelle simulation
REPSIM : reprise d'une ancienne simulation
SPEXEC : superviseur général d'une simulation
EXMD : exécution d'un module (m)
EXAC : exécution d'une action (a)
EXPG : exécution d'un programme (p)
EXIND : exécution d'une indépendance
EXSI : exécution d'un si
EXTQ : exécution d'un tantque
EXSUIV : exécution d'un suivant
EXTPS : exécution d'une condition temporelle (t)
EXDEC : exécution d'une décision (d)
EXBAS : exécution d'un programme (b)
EXDECN : exécution d'une décision (dn)
EXBASN : exécution d'un programme (bn)
SUSPSIM : suspension d'une simulation
TUSIM : annulation d'une simulation
FINSIM : fermeture d'une simulation
TRSIM : trace d'une simulation
TRSIM2 : trace partielle d'une simulation.

3.10. CONCLUSIONS : QUELQUES CONDITIONS SUPPLEMENTAIRES POUR CONSTRUIRE UN VERITABLE PROTOTYPE

Si nous avons évoqué, dans ce chapitre, quelques conditions nécessaires pour construire un prototype, et si nous avons proposé une solution technique pour les remplir, il faut bien constater expérimentalement qu'elles ne sont pas suffisantes.

En effet, pour construire un prototype qui en soit véritablement un, il faut que d'autres exigences soient satisfaites. Elles ne sont pas contradictoires avec MACSI2, mais leur respect est impératif pour que la réalisation d'un prototype s'avère utile.

Examinons les principales.

a) La dernière version du prototype ne doit pas être techniquement beaucoup plus élaborée que la version finale. En particulier, il est dangereux de réaliser, dans le prototype, des commandes de saisie de données qui fonctionnent en temps réel selon un mode conversationnel, alors que,dans la version finale,elles donneront lieu à des traitements en temps différé. Si les délais d'exécution ne sont pas du même ordre de grandeur dans le prototype et dans la version finale, les personnes qui testent le prototype éprouveront une frustration inévitable lorsqu'elles se serviront de la version définitive beaucoup plus fruste.

b) Le jeu d'essai qui est stocké dans la base de données du prototype doit être suffisamment volumineux pour ne pas présenter de différences trop significatives avec la taille des fichiers de la version définitive. Or, la constitution manuelle d'un jeu d'essai de qualité est une tâche longue et coûteuse. Il peut donc être nécessaire de rajouter à la programmation du prototype proprement dit, celle d'un générateur automatique de jeux d'essai qui produise des données en quantité suffisante. Cette contrainte n'est pas toujours surmontable et peut quelquefois rendre impossible la réalisation d'un véritable prototype.

c) Il faut enfin veiller à ce que les personnes qui utiliseront le prototype soient représentatives des véritables utilisateurs de la version finale du projet. La notion d'utilisateur d'une application informatique est floue. Nous pouvons l'illustrer par cette question :

Dans le système de réservation de places d'une compagnie aérienne ou de la SNCF, qui sont les utilisateurs ?

- l'employé qui effectue les réservations devant un terminal ?
- le voyageur qui vient demander un billet ?
- tous les deux ?

Une réponse claire à ce genre de question est préalable à toute conception d'un prototype.

0

0 0

CHAPITRE 4

MACSI3 : UN ESSAI POUR DEFINIR UN LANGAGE D'ANALYSE FONCTIONNELLE

4.1. INTRODUCTION

Les deux versions de MACSI1 présentées aux chapitres précédents ont mis en évidence le souci prioritaire de formaliser les aspects fonctionnels d'une application.

Cependant, dans un souci d'efficacité, nous avons été obligés d'accepter la contrainte consistant à définir artificiellement, à partir d'une description logique des données, une structure Socrate associée. C'est en fonction de cette structure que sont définis les algorithmes (les OPERATEURS de MACSI1, les PROGRAMMES de MACSI2) qui traitent les données.

Cette obligation d'exécuter dans l'ordre d'abord la définition logique des données, ensuite le choix d'une structure Socrate sémantiquement équivalente, enfin la spécification des traitements, nous est apparue contraignante. En effet, en examinant par introspection personnelle ou par une observation du comportement des élèves dans des exercices d'analyse quel est le mécanisme psychologique le plus naturel pendant l'analyse fonctionnelle d'un projet, nous pensons nécessaire la simultanéité des trois démarches suivantes :

- l'analyse des données
- l'analyse des traitements
- l'analyse des procédures administratives.

En fait, avec des associations d'idées, l'analyste spécifie son projet suivant un processus d'analyse, par approximations successives, qui concernent tour à tour les données, les procédures administratives ou les traitements. Il est donc véritablement contraignant d'être obligé d'avoir à réaliser les spécifications complètes de la description des données pour pouvoir commencer à décrire ensuite les traitements.

Par ailleurs, nous avons souvent remarqué, à la fin d'un enseignement du langage Socrate, destiné à des analystes, que nous devions répondre à cette question : "Si ce langage permet évidemment d'exprimer une "bonne part de la sémantique" d'une application, l'obligation de prendre en compte les méthodes d'accès que suppose le choix d'une structure particulière, ne permet pas de distinguer clairement

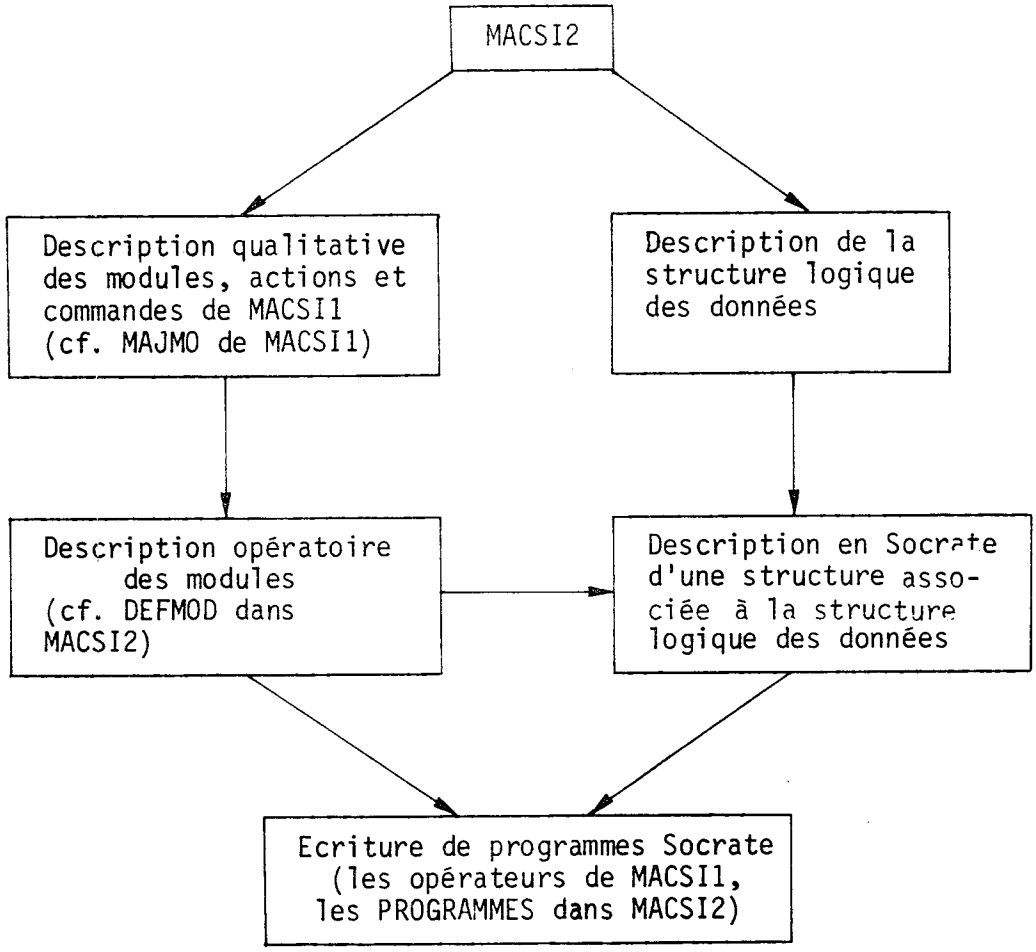
les aspects fonctionnels des aspects organiques : existe-t-il des critères permettant de distinguer ces deux aspects ?"

Il n'est pas facile de répondre en termes précis à cette question.

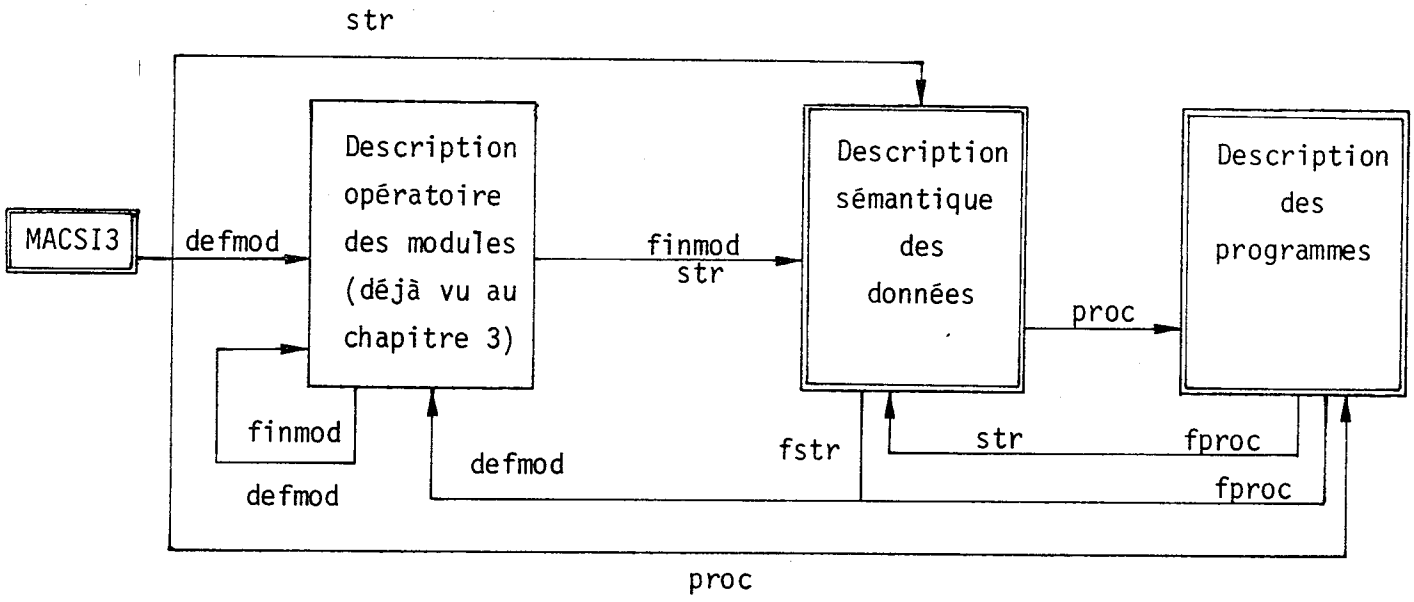
4.2. OBJECTIFS DE MACSI3

Les deux difficultés précédemment exprimées nous amènent donc, après les présentations de MACSI1 et MACSI2, à proposer un formalisme de description purement fonctionnel d'une application qui soit dégagée des contraintes de gestion physique des données qui subsistent dans Socrate et qui permette ainsi de décrire en parallèle les modules, les traitements et les données.

On peut résumer la progression de l'analyse dans MACSI2 par le schéma suivant qui fait apparaître trois étapes principales et rappelle qu'il est impossible, en cas de modifications souhaitées dans la description du projet, de revenir en arrière facilement de la deuxième étape vers la première, ou de la troisième vers la seconde.



Le projet MACSI3 vise à proposer, par opposition, une progression dans l'analyse, affranchie de Socrate et permettant en une seule étape la description fonctionnelle complète d'un projet. Nous pouvons représenter la progression dans l'analyse souhaitée avec MACSI3 par le schéma suivant :



Pour atteindre cet objectif, il faut donc proposer un formalisme de définition des données et des opérateurs qui remplace la description de la structure de données et des programmes écrits en Socrate dans MACSI2. Il faut d'autre part pouvoir à tout instant compléter ou modifier la description des modules, celle des programmes ou celle des données.

4.3. PRINCIPES GENERAUX DE MACSI3

MACSI3 se présente comme un langage permettant donc de définir fonctionnellement un projet.

La structure générale proposée pour le langage est celle représentée par la grammaire suivante :

4.3.1. Structure générale du langage

G1 <MACSI3> → macsi3 <liste spécif fonct> fmacsi

G2 <liste spécif fonct> → <spécif fonct> |
<spécif fonct><liste spécif fonct>

G3 <spécif fonct> → <déf de module> | doop <descr donnée opérateur> fdoop

G4 <déf de module> → defmod <code module><libellé><liste de composants> finmod

| suivi de la liste des règles de grammaire définie dans OASIS pour permettre la description d'un module (page 3.28)

G5 <descr donnée opérateur> → <structure de donnée> | < liste d'opérateur>

G6 <structure de donnée> → str <définition de données> fstr

| partie du langage permettant de décrire une structure sémantique de données selon le modèle présenté dans MACSI1 au § 2.3.5.

< liste opérateur> → <opérateur> | <opérateur><liste opérateur>

G7 < opérateur> → proc <code programme><programme> fproc

| règles de définitions des programmes exprimant la logique des traitements opérant sur les données définies par STR. Evidemment, tout "code de programme" doit être contenu dans une définition de module écrite avec OASIS.

En ne reprenant pas la présentation de OASIS faite au chapitre 3 et que nous conservons intégralement, nous voyons qu'il faut donc définir :

- Le langage de définition sémantique des données en précisant les dérivations possibles de <définition de données> dans G6 : c'est l'objet du § 4.4.

- Le langage de définition de procédures constitué des règles de dérivations de <programme> dans G7 : il sera présenté au § 4.5.

Mais auparavant, examinons quel usage peut être fait de la description d'un projet écrit avec MACSI3.

4.3.2. Utilisation des spécifications écrites avec MACSI3

La définition du langage MACSI3 peut être envisagée dans la seule perspective de présenter un formalisme utilisable pour décrire un problème informatique de gestion, tout comme le formalisme ALGOL permet de décrire un algorithme.

Il est possible aussi de proposer l'exploitation par ordinateur d'une description écrite en MACSI3 pour fournir certaines facilités d'aide à l'analyse. C'est cette deuxième solution que nous avons retenue. Examinons la nature des programmes d'aide à l'analyse qui exploitent un source écrit en MACSI3 et la finalité des résultats qu'ils procurent.

1) - La description des modules sous la commande defmod, sera analysée syntaxiquement par les programmes MODANA et BILMOD présentés au chapitre 3. Ils permettent de vérifier que la description modulaire est syntaxiquement correcte et complète. Nous n'y reviendrons pas.

2) - La description des données et des programmes sera aussi analysée syntaxiquement pour permettre une vérification de la correction syntaxique de la description et une évaluation automatique des correspondances existant entre chaque ensemble, propriété ou relation et les programmes qui opèrent dessus.

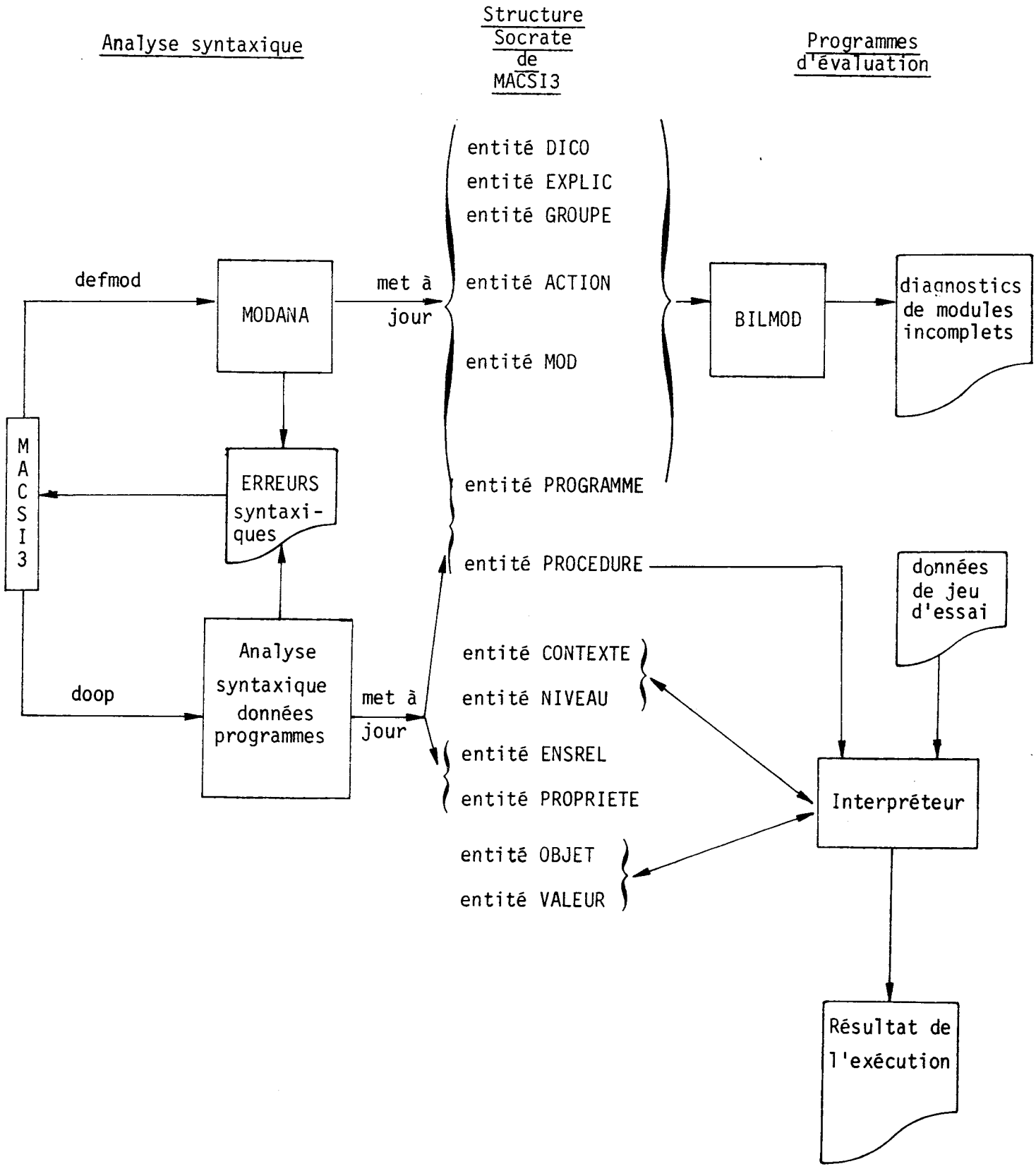
3) - La description des données et programmes doit pouvoir être, de plus, éventuellement exécutée par un interpréteur fonctionnant à partir du pseudo-code produit par l'analyse syntaxique. L'existence de cet interpréteur peut améliorer en effet l'emploi de MACSI3 sur certains points :

   - Il facilitera l'apprentissage initial du langage en autorisant une exécution des spécifications écrites avec ce langage.
   - Il illustrera, en présentant des mécanismes de gestion de l'espace des données, les possibilités d'évolution dynamique du modèle

sémantique des données sans reconfiguration de l'espace de stockage de ces données, possibilités fondamentales qui ne sont pas remplies par Socrate.

- Il permettra enfin d'archiver des jeux d'essais représentatifs des données collectées dans le projet décrit.

Les différents programmes utilisés pour exploiter les spécifications écrites avec MACSI3 et la structure documentaire correspondante peuvent être résumés dans le schéma suivant.


Analyse syntaxique
Structure Socrate de MACSI3
Programmes d'évaluation
defmod
MODANA
met à jour
entité DICO
entité EXPLIC
entité GROUPE
entité ACTION
entité MOD
BILMOD
diagnostics de modules incomplets
MACSI3
ERREURS syntaxiques
entité PROGRAMME
entité PROCEDURE
données de jeu d'essai
doop
Analyse syntaxique données programmes
met à jour
entité CONTEXTE
entité NIVEAU
entité ENSREL
entité PROPRIETE
Interpréteur
entité OBJET
entité VALEUR
Résultat de l'exécution

4.3.3. Principes d'implémentation

L'ensemble des programmes de MACSI3, y compris l'interpréteur, sont écrits en Socrate. Ce choix était dicté par une nécessité de cohérence avec MACSI2, et par la rapidité d'implémentation qu'il procure.

Ce choix délibéré supprimait d'avance tout espoir de performances pour l'exécution des spécifications de programmes exprimées avec MACSI3. Aussi, ce manque évident de performances excluait-il toute perspective de simulation avec cet outil : c'est pourquoi nous ne retenons pas dans MACSI3 les possibilités de simulation présentées au chapitre 3 pour MACSI2. Par ailleurs, une des priorités retenue dans le cadre du projet MACSI3 était, avant tout, d'étudier d'un point de vue ergonomique l'intérêt d'un tel langage pour améliorer la cohérence des études fonctionnelles d'un projet informatique. Par conséquent, comme nous souhaitions surtout que toute modification ou extension de la grammaire de MACSI3 puisse être prise en compte par des modifications correspondantes faciles de l'analyseur syntaxique et de l'interprêteur, nous avons adopté une technique d'écriture de ceux-ci largement inspirée de celle adoptée par J.R. ABRIAL et J.C. FAVRE dans le projet SOCRATEP [18] et que nous avons déjà utilisée dans l'implémentation du langage OASIS (§ 3.8) : nous ne reviendrons pas sur ses caractéristiques.

Nous présenterons successivement :

- le sous-ensemble de MACSI3 permettant de décrire la structure sémantique des données en précisant :
  - . la structure de l'espace de stockage des données
  - . la grammaire du langage
  - . les instructions permettant certains diagnostics de cohérence
- le sous-ensemble de MACSI3 utilisé pour décrire la logique des programmes en indiquant les caractéristiques essentielles de l'analyse syntaxique.

4.4. LANGAGE DE DESCRIPTION DES DONNEES.

Ce langage donne un moyen d'exprimer, dans le cadre d'un projet, la logique des données selon le modèle présenté dans MACSI1 au § 2.3.5.. Ce langage ne fait donc intervenir aucune notion d'accès à des structures physiques de données.

4.4.1. Structure de l'espace de stockage des données

Il est constitué des quatre entités suivantes :

```
entité 5000 VALEUR
       début VFCT de 1 à 500
       |     VMOT mot rapide
       |     VNUM de 0 à 100000
       |     VTEX référence un EXPLIC
       |     VOBJ de 0 à 2000
       fin

entité 500 PROPRIETE
       début CDPROP mot rapide
       |     TYVAL de 1 à 10
       |     DEQUOI de 0 à 100
       |     VALASS inverse tout VALEUR
       |     DEF2 référence un EXPLIC
       fin

entité 2000 OBJET
       début OBJENS de 1 à 100
       |     OBJ1 de 1 à 2000
       |     OBJ2 de 1 à 2000
       |     VALOBJ inverse tout  VALEUR
       |     OBJINV inverse tout  OBJET
       fin
```

```
entité 100 ENSREL
       début CDENS mot rapide
       |     DEF1 référence un EXPLIC
       |     PROP inverse tout PROPRIETE
       |     CONTENU inverse tout OBJET
       |     TY de 1 à 2
       |     ARG1 de 1 à 100
       |     ARG2 de 1 à 100
       fin
```

a) Chaque ensemble ou chaque relation sera décrit comme une réalisation de l'entité (ENSREL).

Chaque ensemble ou relation aura un identificateur qui est son (CDENS). Sa définition sera stockée dans (DEF1). Que ce soit un ensemble (TY=1) ou une relation (TY=2), il faudra indiquer toutes ses propriétés dans (PROP). Quand il s'agit d'une relation, ARG1 et ARG2 indiquent les numéros d'occurrences des réalisations correspondant aux ensembles arguments de la relation.

Mais, à chaque ensemble correspond, à un instant donné, tous les objets qui en font partie : un certain nombre d'entre eux peut être identifié dans (CONTENU). Pour cela, MACSI3 gère un espace des objets qui est matérialisé par l'entité (OBJET) : à chaque réalisation de l'entité OBJET, correspond un objet, distinct de tous les autres.

Restriction fondamentale : Chaque objet appartient à un ensemble et un seul (OBJENS) repéré par son numéro d'occurrence dans l'entité ENSREL [(1)]. Aussi, quand un élément ℓ_i peut être sémantiquement considéré comme étant un objet de l'ensemble E_α, un objet de l'ensemble E_β et un objet de l'ensemble E_γ, il sera généré trois réalisations $\ell_{i\alpha}$ $\ell_{i\beta}$ et $\ell_{i\gamma}$ de l'entité OBJET, chacune rattachée à un ensemble différent et n'ayant entre elles aucune relation. Pour considérer que $\ell_{i\alpha}$ $\ell_{i\beta}$ et $\ell_{i\gamma}$ sont en fait le même être, il faut poser entre les ensembles E_α E_β et E_γ des relations sémantiquement signifiantes.

---

(1) - A la différence du projet SOCRATEP (réf. [18]) où, si un objet appartient aussi à un espace unique des objets, il peut être affecté dynamiquement à plusieurs ensembles ou relations.

b) Chaque propriété (PROPRIETE), identifiée par son code (CDPROP), est rattachée à l'ensemble unique sur lequel elle est définie par (DEQUOI), numéro d'occurrence de cet ensemble ou de cette relation dans l'entité ENSREL. Elle est définie par un texte (DEF2). Les valeurs de cette propriété, quel que soit son type (TYPVAL), sont repérées dans l'espace unique des valeurs, correspondant à l'entité (VALEUR), par le fichier inverse VALASS.

c) Chaque objet dans l'entité (OBJET) est rattaché à :

- l'ensemble auquel il appartient (OBJENS)
- les valeurs des propriétés associées à cet objet (VALOBJ).

Si cet objet correspond à une relation binaire, il s'agit, en fait, d'un couple de deux constituants qui sont repérés par (OBJ1) et (OBJ2).

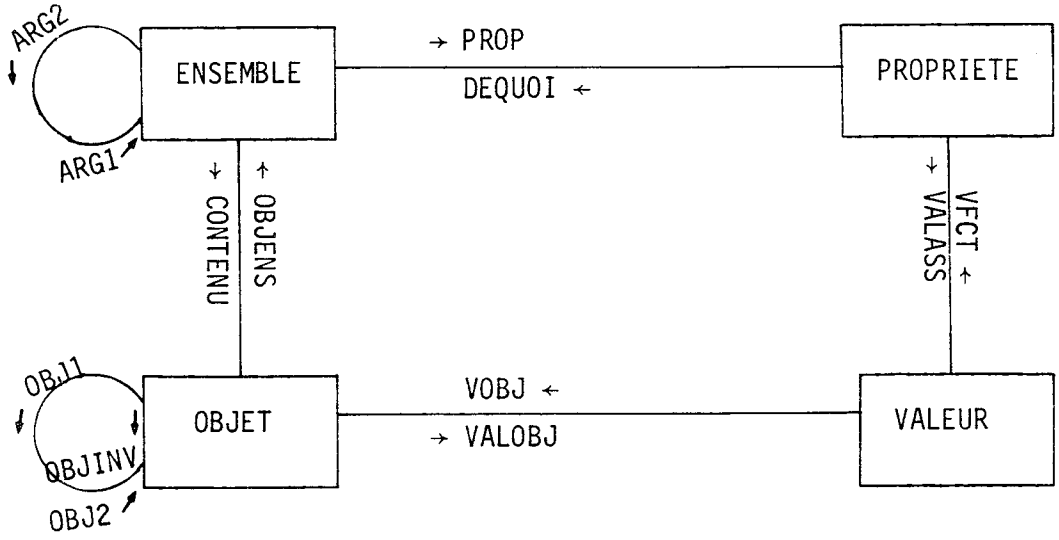
Réciproquement, chaque objet peut intervenir comme élément dans des couples appartenant à diverses relations : chacun de ces couples est repéré dans (OBJINV).

d) Enfin, chaque valeur (VALEUR) correspond à une certaine propriété et une seule (VFCT) : selon le type (TYPVAL) de cette propriété, on a :

- si TYPVAL = 1, c'est un mot stocké dans (VMOT)
- si TYPVAL = 2, c'est un numérique stocké dans (VNUM)
- si TYPVAL = 3, c'est un texte repéré par (VTEX) et stocké dans (EXPLIC).

L'objet auquel est associée cette valeur est repéré par (VOBJ), numéro d'occurrence de cet objet dans l'entité OBJET.

Ainsi, les relations entre les espaces de propriétés, de valeurs, d'objets et d'ensembles peuvent être résumées dans le schéma suivant.



4.4.2. Grammaire du langage de description des données

Les règles de description des ensembles, de leurs propriétés et de leurs relations sont définies ainsi :

D1 <définition de données> → <manip ens>|<manip ens><définition de données>
D2 <manip ens> → <création-ens>|
D3 → <modif-ens>|
D4 → <suppression>|
D5 → <édition-ens>
D6 <création-ens> → ce <cdens>*<def1>*<structure>|
D7 → cr <cdens>(<cdens> <cdens>) * <def1> * <structure>
D8 <structure> → <cr-prop>|<cr-prop><structure>
D9 <cr-prop> → mot <cdprop> * <def2> *|
D10 num <cdprop> * <def2> *|
D11 tx <cdprop> * <def2> *
D12 <modif-ens> → me <cdens><structure>
D13 <édition-ens> → cdens <cdens>
D14 <suppression> → situer< cdens>|t <cdens> |
D15 t <cdprop>

Commentaires

Une définition des données est une suite de manipulation d'ensembles (règle D1). Chaque manipulation est soit une création (D2), c'est-à-dire la première définition d'un ensemble, soit une modification d'un ensemble

(D3), soit une suppression (D4). Enfin, on peut demander l'édition (D5) de la définition d'un ensemble.

Création d'un ensemble (D6 à D11)

Après le mot réservé "ce", on indique le code de l'ensemble, sa définition et sa structure, c'est-à-dire la liste de ses propriétés.

S'il s'agit de créer une relation (D7), on indiquera entre parenthèses ses deux arguments, le mot réservé étant alors "cr".

Chaque propriété d'un ensemble est d'abord indiquée par son type, ensuite son code, enfin par le texte de sa définition.

Exemple de structure de données (concernant toujours la gestion d'un congrès)

```
STR
    ce PERSONNE * Personnes ayant reçu une publicité et un bulletin d'ins-
                  cription à un congrès *
            mot NOM * nom de la personne *
            tx  ADRESSE * adresse où doit lui parvenir les publications du
                          congrès *
            mot SEXE * sans commentaires *
            num AGE  * âge du congressiste *

    ce SESSION * sessions dont se compose le congrès *
            mot CDSESS * code de la session *
            tx TITRE * titre de la session *
            num DR * droit à acquitter pour participer à cette session *

    cr INSCRIPTION (PERSONNE SESSION) * inscription au titre d'une session *
            mot DATE * date de l'inscription *
            num PAYE * montant payé au titre de cette session : il peut être
                       inférieur ou supérieur au droit correspondant *
FSTR
```

Cette description pourrait, parmi les nombreuses structures Socrate sémantiquement équivalentes, correspondre à la suivante :

```
entité PERSONNE
        début NOM mot
        |       ADRESSE texte
        |       INSCRIPT inverse tout SESSION
        |       SEXE mot
        |       AGE de 1 à 80
        fin

entité SESSION
        début CDSESS mot
        |       TITRE texte
        |       DR de 50 à 300
        |
        |       entité INSCRIPT
        |               début QUI référence une PERSONNE
        |               |       PAYE de 1 à 1000
        |               |       DATE mot
        |               fin
        fin
```

4.4.3. Modifications de la structure sémantique des données et diagnostics de cohérence

Modification d'un ensemble (DI 2)

Il est possible de rajouter des propriétés à un ensemble existant même lorsque des objets de cet ensemble ont déjà été créés, ainsi que des valeurs des propriétés déjà définies pour ces objets.

Exemple : Supposons qu'à la fin de l'exemple précédent, nous ayons ensuite écrit des programmes de création d'une inscription ou d'une personne et que, pour les tester, nous ayons généré comme jeu d'essai quelques personnes et quelques inscriptions. On pourrait alors revenir dynamiquement sous STR pour faire la modification suivante :

```
STR
   me INSCRIPTION
      texte CHEQUE * numéro et libellé de la banque associés au chèque
                     versé pour le montant payé *
FSTR
```

Suppression d'une propriété (DI 5)

L'instruction t <cdprop> a pour effet :

- suppression de la réalisation de l'entité "propriété" ayant ce code
- simultanément, suppression de toutes les valeurs associées à cette propriété dans VALASS.

De plus, est mis à jour le pointeur PROP de l'ensemble auquel était rattachée cette propriété ainsi que, pour chaque objet existant de cet ensemble, le VALOBJ de cet objet.

Suppression d'un ensemble (DI 4)

C'est une opération extrêmement complexe dont l'exécution doit être réalisée par deux instructions distinctes.

1°) Diagnostic des implications de cette suppression si elle est effectivement réalisée. Ce diagnostic est demandé par l'instruction "situer <cdens>". Il édite pour l'analyste une évaluation des conséquences de cette suppression si elle est exécutée.

2°) Si l'analyste, après le diagnostic, est confirmé dans son intention de supprimer l'ensemble, il en commande l'exécution par l'instruction "t<cdens>".

L'exécution de cette instruction est effectuée récursivement comme suit :

- Pour toute propriété de l'ensemble repérée dans PROP, exécution implicite d'une opération équivalente à t<cdprop> pour cette propriété.

- Pour toute relation dans laquelle cet ensemble est soit ARG1, soit ARG2, exécution implicite d'une opération équivalente à t<cdens> pour cette relation.

- Suppression de tous les objets associés à l'ensemble par CONTENU.

Donc, le diagnostic de la suppression d'un ensemble E doit analyser quelles sont toutes les relations R_i dont E est argument et les propriétés de ces relations ; il doit aussi rechercher tous les programmes (cf. infra § 4.5.3.) qui travaillent sur E, sur les propriétés de E, sur les R_i et sur les propriétés de R_i : tous ces opérateurs devront donc être déclassés.

Edition d'un ensemble (DI 3)

L'instruction edens<cdens> a pour effet de récapituler à un instant donné quelles sont toutes les propriétés d'un ensemble, ainsi que celles de toutes les relations dans lesquelles il intervient.

4.5. LANGAGE DE DESCRIPTION DES PROGRAMMES

Nous avons défini ce langage pour qu'il reprenne, dans toute la mesure du possible, les caractéristiques du langage de requête Socrate. Ce choix est inspiré par les trois considérations suivantes :

- Pour tous ceux qui ont déjà une pratique de Socrate (en particulier les analystes qui voudraient modifier MACSI), il est souhaitable de rester dans le cadre de notations qu'ils connaissent.

- Le langage de requête Socrate possède des qualités de concision et de clarté suffisantes pour exprimer clairement des schémas de programmes, qualités que nous avons exploitées dans MACSI1. Il est donc souhaitable de les conserver.

- La comparaison entre le langage présenté ici et Socrate permet de faire apparaître, par leurs différences, ce qui, dans Socrate, correspond aux choix organiques faits en précisant les méthodes d'accès à la base de données.

4.5.1. Grammaire du langage de requêtes

R1 <opérateur> → proc <code programme><programme> fproc (cf. règle G7)
→ proc <code programme><liste de par><programme> fproc

R2 <programme> → <liste req>

R3 <liste req> → <Req>|<Req><liste req>

R4 <Req> → <Req élém>|

R5 <Req groupe>|

R6 <Debranchement>|

R7 <commentaire>

R8 <Commentaire> → /* <liste de chaîne>* /

R9 <Req élém> → <citation>|
<mise à jour>|<suppression>|<génération>
<interrogation>|
<appel>|<compilation>|
<édition>

R10 <Débranchement> → Refaire

R11 → Sortie

R12 → Suivant

R13 → Retour < liste de param >

R14 <Requête groupée> → pour <cite><liste req> fin|

R15 faire <liste req> fin|

R16 si <condition> alors <liste req> [sinon<liste req>] fin

R18 <Citation> → soit <cite>

R19 <cite> → <quantificateur><cdens> $X_i[X_jX_k]$[<filtre>]

R20 <quantificateur> → un|une|tout|toute

R21 <filtre> → ayant <condition> ;

R22 <condition> → <cond simple>|<cond simple><op bool><condition>

R23 <cond simple> → <citp><opc><citp>|

R24 <cdprop><opc><citp>|

R25 <cdprop><opc><chaîne>|

R26 <cdprop><opc><nb>|

R27 <citm><opc><chaîne>|

R28 <citn><opc><nb>|

R29 <citn> → <citc-n>|Y_i

R30 <citm> → <citc-m>|Z_i

R31 <citp> → <citm>|<citn>|<citc-t>

R32
<citc-m> → <cdprop> de X_i /* pour des propriétés de type mot */
<citc-n> → <cdprop> de X_i /* pour des propriétés de type numérique */
<citc-t> → <cdprop> de X_i /* pour des propriétés de type texte */

R33 <opbool> → ou | et

R34 <opc> → = | ¬= | > | < | >= | <=

R35 <mise à jour> → m <citp> = ext

R36 m <citp> : <valeur>

R37 m X_i = X_j

R38 m <citm> = {<citm>|<chaîne>| nul|u}

R39 m <citn> = <expression>

R40 m <citc-t> = <citc-t>

R41 <expression> → <citn>|<nb>|<citn><opa><expression>|<nb><opa><expression>

R42 <opa> → * | / | + | -

R43 <suppression> → t <cite> | t X_i

R44 <génération> → g <cite>

R45 <interrogation> → i [(<num><num>)] <citp> | i [(<num><num>)] <chaîne>

R46 <édition> → edit <nom-proc>

R47 <compilation> → compil <nom-proc>

R48 <appel> → call <nom-proc> [<liste param>]

R49 <liste param> → <parm> | <parm><liste param>

R50 <parm> → X_i | Z_i | Y_i | <nb>

R51 <liste de par> → <cite>|<cite><liste de par>

Règles complémentaires

<cdens> → nom d'ensemble ou de relation définie sous STR

<cdprop> → code de propriété d'un ensemble ou d'une relation définie sous STR

<nom proc> → <chaîne>/*nom de programme*/

<valeur> → <chaîne>|<num>|<texte>

<chaîne> → 16 caractères alphanumériques au plus

<texte> → <ligne><RC>**|<ligne><RC><texte>

<ligne> → 60 caractères au maximum

<num> → entier positif.

4.5.2. Explications complémentaires sur le langage de requête

Nous donnerons ici quelques commentaires sur les seules règles qui diffèrent nettement du langage Socrate spécifié dans la référence [37].

Les différences sont, du reste, dues essentiellement à la définition des relations en tant qu'ensembles particuliers, alors que dans Socrate, elles sont en général des propriétés.

Règles R18, R19 - Soit <cite>

Quand la citation d'ensemble porte sur une relation, les propriétés qui figurent dans le filtre peuvent être soit celles de la relation, soit celles de ses deux arguments.

Exemple : On peut écrire :

Soit toute inscription X1 X2 X3 ayant NOM de X2 ≠ u et PAYE de X1 = DROIT de X3 ; call QUITTANCE X1

Par contre, il est impossible de pouvoir imbriquer les citations d'ensembles ou de relation dans un filtre.

Règles R23 à R32 - Citation de propriété <citp>

Les citations de propriétés peuvent être de trois types : des mots (<citm>), des textes (<citc-t>) ou numériques (<citn>) : il y a toujours vérification que, dans une condition simple, les deux termes de l'opérateur de comparaison sont de même type.

Règle R36 - m<citp> : <valeur>

Lorsque l'on compare une propriété à une valeur, le signe ":", qui signifie une égalité, a été introduit pour ne pas avoir d'ambiguïté d'interprétation avec les règles R38, R39 et R40 où le signe "=" est obligatoirement suivi d'une citation. Ceci permet d'éviter des délimiteurs pour représenter une valeur directe.

Exemple : Les deux cas suivants permettent d'illustrer cette distinction.

1er cas

g une session X1 m TITRE de X1 : Z1

Alors la valeur de titre sera Z1 (ce qui n'est pas très significatif !), mais on aurait pu écrire m TITRE de X1 : colloque sur les bases de données **

2ème cas

g une session X1 m TITRE de X1 = Z1

Alors la valeur de titre sera le contenu du mot Z1.

Règles R1, R46, R47 - Compilation immédiate ou différée

L'instruction "proc" ... "fproc" (R1) est interprétée comme une analyse syntaxique immédiatement exécutée dès la requête "fproc". S'il existe déjà une procédure ayant ce code, elle est tuée et remplacée par la nouvelle.

Si l'on veut modifier une procédure dont le source est déjà catalogué, on pourra la modifier avec l'éditeur (identique à celui de Socrate) appelé par "EDIT" (R46). Lorsque la compilation sera à nouveau souhaitée, on la commandera par "COMPIL" (R47).

Enfin, l'exécution d'un programme compilé est lancée par la commande <nom-proc> ?

4.5.3. Quelques précisions sur l'analyse syntaxique

Les différentes entités nécessaires pour l'analyse syntaxique et l'interprétation sont :

entité PROGRAMME dans laquelle figure le source
entité PROCEDURE qui permet de stocker le code produit par la compilation
entité ROUTINE où sont stockées toutes les procédures Socrate d'analyse syntaxique et d'exécution du code généré comme dans MACSI2 (§ 3.8.1 page 3.34)
entité NIVEAU qui est une pile de manoeuvre pour gérer les blocs à l'exécution
entité CONTEXT dans laquelle sera géré le contexte de chaque instruction à l'exécution (elle joue le même rôle de l'entité CONTEXTE

dans MACSI2 bien que sa structure soit différente).

La structure de ces différentes entités est la suivante :

entité 200 PROGRAMME

```
début CDPROC mot rapide
|     SOURCE texte 100
fin
```

Dans l'entité PROGRAMME, chaque programme déclaré préalablement dans un module est identifié par son code (CDPROC). Son source (SOURCE) est stocké dès la fin de la définition du programme annoncé par le mot-clef "fproc".

entité 200 ROUTINE

```
début TYPRT mot rapide
|     PANA de 1 à 1000
|     NIVQT de 1 à 100
|     PINT de 1 à 1000
fin.
```

Comme dans MACSI2, toutes les procédures de compilation et d'interprétation du code de MACSI3 sont enregistrées dans (ROUTINE). L'analyse syntaxique et l'interprétation se faisant par appel associatif de la routine correspondant au mot-clé qui identifie la règle en cours d'analyse ou d'exécution, ce mot-clé est stocké dans (TYPRT) ; (PANA) donne le code de la procédure d'analyse syntaxique, (PINT) indique le code de celle utilisée pour l'exécution. (NIVRT) indique le niveau d'appel de ce type de règle.

Le code généré est stocké dans l'entité (PROCEDURE). Un programme compilé de l'entité (PROGRAGRAMME), donne une réalisation de l'entité (PROCEDURE) : ils ont même code d'identification (CDPROC).

Nous ne présenterons pas pour MACSI3 les mécanismes d'analyse syntaxique et d'exécution du pseudo-code généré : ils sont identiques à ceux de MACSI2 déjà largement expliqués.

Cependant, il paraît souhaitable de détailler l'entité PROCEDURE pour voir comment l'analyse syntaxique produit des renseignements permettant certains diagnostics sur la cohérence du source écrit avec MACSI3.

```
entité 500 PROCEDURE
        début CDPROC mot rapide
        |   entité 1000 INSTRUCTION
        |         début TYINST référence un ROUTINE
        |         |     FORME de 1 à 100
        |         |     IND1 de 1 à 100
        |         |     IND2 de 1 à 100
        |         |     IND3 de 1 à 100
        |         |     PROP référence un PROPRIETE
        |         |     V1MOT mot
        |         |     V1NUM de 0 à 100000
        |         |   entité PARAM
        |         |        début TYPARM de 1 à 100
        |         |        |     INDPARM de 1 à 10000
        |         |        fin
        |         |   SUIV1 référence un INSTRUCTION de un PROCEDURE
        |         |   SUIV2 référence un INSTRUCTION de un PROCEDURE
        |         |   SUIV3 référence un INSTRUCTION de un PROCEDURE
        |         |   CHAINAGE de 1 à 1000
        |         fin
        |           MFCT inverse tout PROPRIETE
        |           IFCT inverse tout PROPRIETE
        |           GENS inverse tout ENSEMBLE
        |           LENS inverse tout ENSEMBLE
        |           TENS inverse tout ENSEMBLE
        |           APPEL inverse tout PROGRAMME
        fin
```

Chaque instruction du langage source, une fois analysée et reconnue correcte, provoque la génération d'une réalisation de l'entité (INSTRUCTION).

(TYINST) identifie la (ROUTINE) correspondante : à l'exécution, on appellera PINT de TYINST.

(FORME) permet d'identifier la nature de la citation = Xi, Yi, Zi, mot, numérique, texte, citation d'ensemble, etc ....

(IND1) (IND2) (IND3) donnent les indices des Xi manipulés par l'instruction.

(PROP) identifie la propriété qui est concernée par l'instruction.

(V1MOT) et (V1NUM) permettent de stocker les valeurs directes de type mot ou numérique qui peuvent apparaître dans une instruction de mise à jour (cf. règle R36 du langage).

Dans l'entité (PARM) sont identifiés les paramètres pour les instructions "CALL" (règle R48) et "RETOUR" (règle R13).

(SUIV1) (SUIV2) (SUIV3) identifient les instructions suivantes de l'instruction courante.

Remarques :

1) Au fur et à mesure que chaque instruction est décodée, on met à jour les fichiers MFCT, IFCT, GENS, TENS, LENS qui identifient les propriétés et ensembles concernés par l'instruction.

MFCT = liste des propriétés dont des valeurs sont mises à jour : elles apparaissent dans des parties gauches d'instructions de mise à jour (règles R35, R36, R38, R39 et R40).

IFCT = liste des propriétés dont les valeurs sont lues : elles apparaissent dans des citations de propriétés.

GENS = liste des ensembles pour lesquels des objets sont générés (règle R44).

TENS = liste des ensembles pour lesquels des objets sont supprimés (règle R43).

LENS = liste des ensembles auxquels on accède dans une citation d'ensemble.

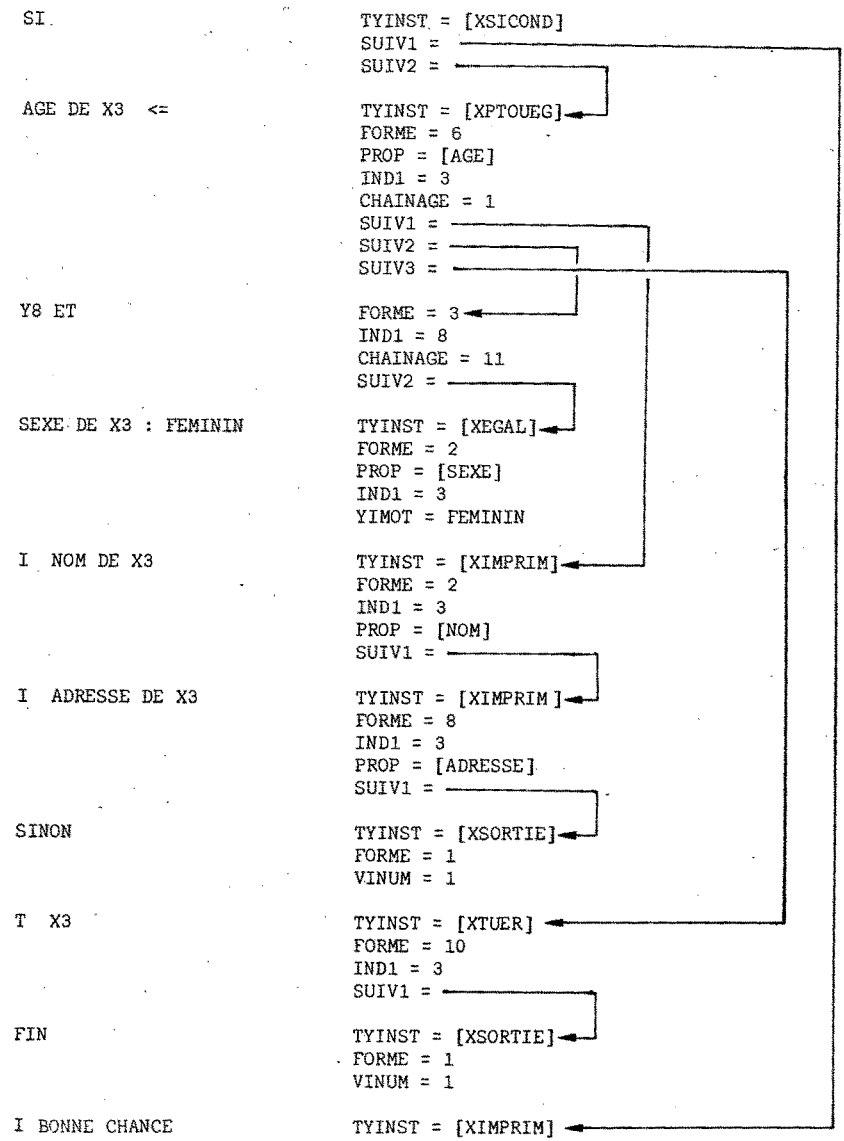
Ce sont ces fichiers inverses qui permettent d'effectuer le diagnostic sur les conséquences de la suppression d'un ensemble (cf. supra règle D14).

2) De même en décodant les instructions CALL (règle R48), on met à jour le fichier APPEL de tous les programmes appelés par un programme donné.

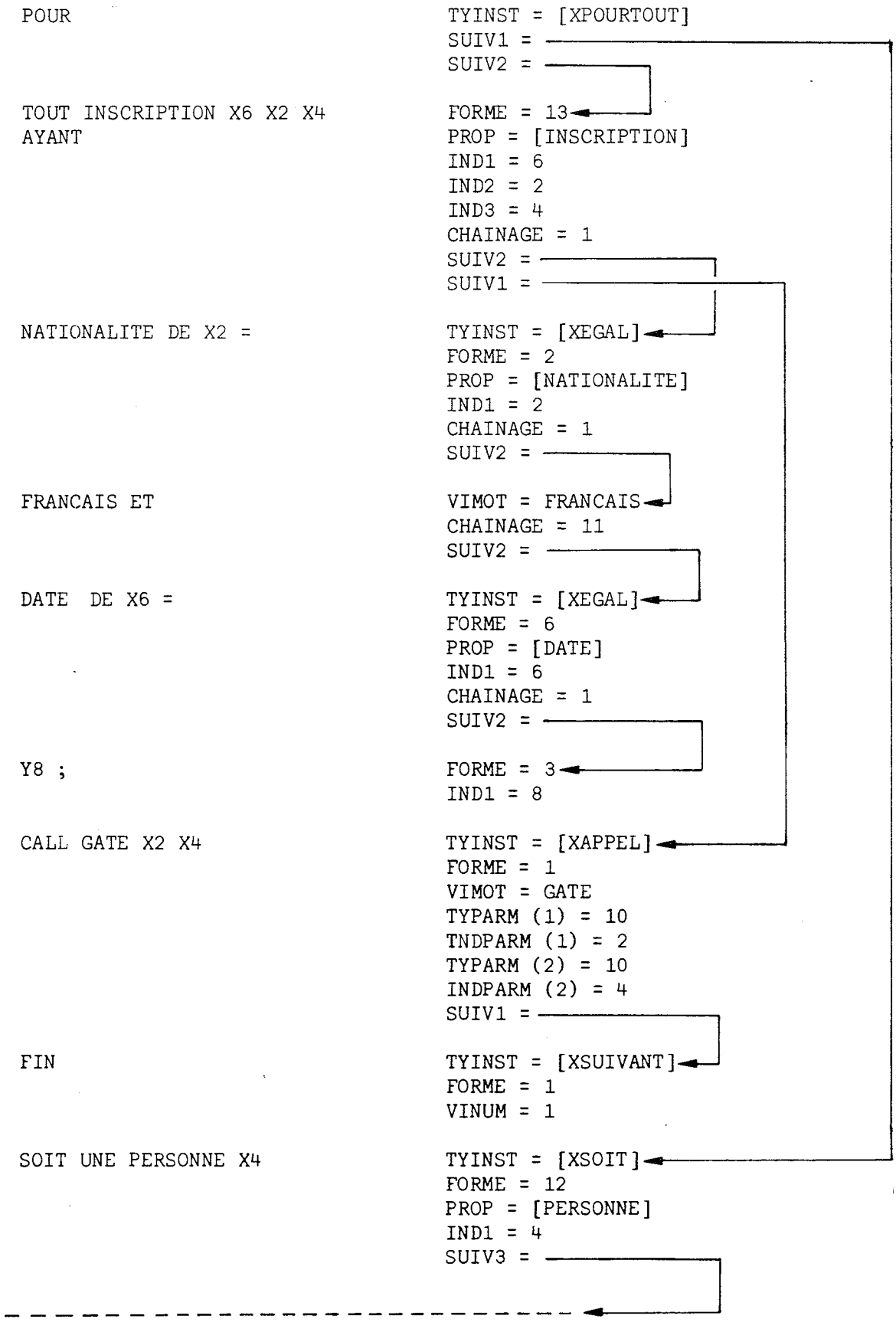
3) On a ainsi, comme sous-produit de l'analyse syntaxique, un bilan, en permanence à jour, des correspondances entre programmes et données, comme entre programmes appelés et programmes appelants. Le lecteur aura évidemment noté que ce bilan effectué sur un source écrit en MACSI3 pourrait tout aussi bien être fait sur un source écrit en Socrate.

Exemple : Nous donnons deux exemples du code généré dans (PROCEDURE) à partir de deux séquences d'instructions définies sur l'exemple de structure de données du § 4.4.2..

Exemple 1


SI
TYINST = [XSICOND]
SUIV1 =
SUIV2 =
AGE DE X3 <=
TYINST = [XPTOUEG]
FORME = 6
PROP = [AGE]
IND1 = 3
CHAINAGE = 1
SUIV1 =
SUIV2 =
SUIV3 =
Y8 ET
FORME = 3
IND1 = 8
CHAINAGE = 11
SUIV2 =
SEXE DE X3 : FEMININ
TYINST = [XEGAL]
FORME = 2
PROP = [SEXE]
IND1 = 3
YIMOT = FEMININ
I NOM DE X3
TYINST = [XIMPRIM]
FORME = 2
IND1 = 3
PROP = [NOM]
SUIV1 =
I ADRESSE DE X3
TYINST = [XIMPRIM]
FORME = 8
IND1 = 3
PROP = [ADRESSE]
SUIV1 =
SINON
TYINST = [XSORTIE]
FORME = 1
VINUM = 1
T X3
TYINST = [XTUER]
FORME = 10
IND1 = 3
SUIV1 =
FIN
TYINST = [XSORTIE]
FORME = 1
VINUM = 1
I BONNE CHANCE
TYINST = [XIMPRIM]
FORME = 1

Exemple 2


POUR
TYINST = [XPOURTOUT]
SUIV1 =
SUIV2 =
TOUT INSCRIPTION X6 X2 X4
AYANT
FORME = 13
PROP = [INSCRIPTION]
IND1 = 6
IND2 = 2
IND3 = 4
CHAINAGE = 1
SUIV2 =
SUIV1 =
NATIONALITE DE X2 =
TYINST = [XEGAL]
FORME = 2
PROP = [NATIONALITE]
IND1 = 2
CHAINAGE = 1
SUIV2 =
FRANCAIS ET
VIMOT = FRANCAIS
CHAINAGE = 11
SUIV2 =
DATE DE X6 =
TYINST = [XEGAL]
FORME = 6
PROP = [DATE]
IND1 = 6
CHAINAGE = 1
SUIV2 =
Y8 ;
FORME = 3
IND1 = 8
CALL GATE X2 X4
TYINST = [XAPPEL]
FORME = 1
VIMOT = GATE
TYPARM (1) = 10
TNDPARM (1) = 2
TYPARM (2) = 10
INDPARM (2) = 4
SUIV1 =
FIN
TYINST = [XSUIVANT]
FORME = 1
VINUM = 1
SOIT UNE PERSONNE X4
TYINST = [XSOIT]
FORME = 12
PROP = [PERSONNE]
IND1 = 4
SUIV3 =

4.6. CONCLUSION

Le langage MACSI3, tel qu'il vient d'être défini, exprime, en fait, les possibilités indispensables que devrait offrir un système de base de données de haut niveau facilitant au maximum la description et la simulation d'un projet informatique de gestion.

Il reste à utiliser MACSI3 de manière intensive, pour exprimer la logique de divers projets, de façon à dégager quelles modifications ou extensions seraient intéressantes. Certaines sont, du reste, dès maintenant prévisibles. Elles concerneraient notamment :

- la possibilité, à partir d'un ensemble, de définir, par une relation d'équivalence, des sous-ensembles de celui-ci, ayant mêmes propriétés que lui.

- une plus grande diversité dans les types de données.

- la définition complète des actions à l'intérieur de DEFMOD.

Quelles que soient ces extensions, il faut bien admettre que les spécifications définitives de MACSI3 seront, sans doute, assez proches d'autres langages, développés dans le domaine de l'intelligence artificielle, qui ont des objectifs communs avec MACSI3. Entre autres, signalons :

- l'expression d'algorithmes indépendamment de leurs modalités d'exécution par la machine.

- la description ensembliste des données et des possibilités de recherche associative sur celles-ci.

- des ressources pour réorganiser l'espace physique des données sans modifications correspondantes des algorithmes.

- des facilités pour effectuer des retours-arrière dans les spécifications, une fois analysés les résultats des simulations exécutées.

Constatons alors que le développement de ces langages pose à terme un problème considérable de formation des informaticiens et des analystes. L'utilisation de ces langages supposera, en effet, une connaissance de plus en plus approfondie de la logique mathématique : ceci nécessitera, à notre avis, une transformation assez importante des programmes pédagogiques dans les filières de formation à l'informatique. Nous en voulons, pour preuve, les usages faits aujourd'hui du logiciel Socrate : considéré au départ comme un moyen particulièrement efficace d'expression et d'exécution de programmes dans le domaine de la gestion, ce système de base de données semble, à l'expérience, offrir cette possibilité aux seuls ingénieurs ayant bénéficié d'une formation théorique importante dans les domaines de la logique, de l'algèbre et des techniques de structuration des données.

0

0 0

CHAPITRE 5

BILAN ET PERSPECTIVES

Au terme de cette présentation du projet MACSI, peut-être pouvons-nous oser évoquer les satisfactions obtenues, les nouveaux efforts d'amélioration à entreprendre et les progrès espérés à moyen terme dans le cadre de ce projet.

Satisfaction,tout d'abord,d'avoir pu dégager quelques normes permettant de définir tout formalisme préalable à la conception d'un système informatique de gestion, conception pour laquelle la pluridisciplinarité de l'équipe de réalisation comme celle des critères d'évaluation est une nécessité. L'analyse d'un projet informatique, dans le domaine de gestion, fait appel en effet à des concepts et des méthodes qui relèvent de la sociologie,de l'organisation de la sémiotique et de la psychologie de la perception, du droit, de l'ingéniérie, et évidemment aussi des techniques de gestion et de l'informatique. Pour fédérer harmonieusement des hommes ayant des compétences diverses dans ces différentes spécialités, il faut pouvoir leur proposer des règles leur permettant de construire eux-mêmes un langage qui leur soit commun. La flexibilité recherchée dans MACSI1, l'effort de synthèse de MACSI3 offrent des possibilités de construction d'un langage d'analyse qui laissent à la liberté de chaque équipe réalisatrice d'un projet, le soin de préciser son propre modèle de description compte tenu de ses contraintes spécifiques.

Il n'en reste pas moins que les résultats obtenus seraient partiellement vains s'ils n'étaient pas prolongés par de nouveaux efforts de recherche ou de développement. Un certain nombre de travaux nous paraissent prioritaires dont certains sont déjà démarrés :

= A partir de MACSI1, il est possible de mettre au point un cours d'analyse, assisté par ordinateur. Celui-ci serait fondé sur la réalisation d'un superviseur conversationnel dont le fonctionnement serait piloté par un fichier de règles de description. Ce cours apporterait, dans la pédagogie de l'analyse, une rigueur conceptuelle qui fait jusqu'ici sévèrement défaut.

= Il reste, à partir de MACSI2, à étudier complètement quels peuvent être les algorithmes qui permettent à partir du prototype d'un projet, une conception partiellement automatique de sa version définitive, compte tenu de certaines hypothèses sur les choix techniques de réalisation.

= Enfin, les difficultés rencontrées avec MACSI2 pour réaliser des retours-arrière au niveau de la conception et de la simulation d'un projet nous amènent à souhaiter un langage de haut niveau qui aurait des performances suffisantes, tout en offrant une indépendance aussi large que possible vis-à-vis des méthodes d'accès aux structures physiques de données. Certes, le langage MACSI3 n'a aucune implémentation efficace aujourd'hui, mais il représente un cahier des charges précis des possibilités attendues d'un logiciel utilisable pour décrire et simuler un projet informatique de gestion.

Il nous faudra surtout sauvegarder, au milieu de toutes ces recherches techniques possibles, un certain temps pour la pratique. Elle est le seul moyen d'expérience permettant de mieux comprendre quelles sont les causes profondes de ce qui est, le plus fréquemment, à l'origine d'un échec dans un projet informatique : le manque de dialogue entre tous les hommes qu'il concerne. Provient-il d'une difficulté à prévoir, d'une absence de langage commun, ou de certains manques de formation ? Témoigne-t-il d'un abus de pouvoir, de quelques inquiétudes viscérales face à l'innovation, d'une résistance passive devant une forme d'impérialisme technique, d'un refus de la centralisation ou d'une absence de convivialité de l'outil ?

Il reste à définir une séméiologie des systèmes d'information.

0

0 0

BIBLIOGRAPHIE

[1] J.R. ABRIAL

Projet Socrate.

Cours Avancé sur l'Architecture des Systèmes.
Alpe d'Huez - Décembre 1972.

[2] J.R. ABRIAL, J.P. CAHEN, J.C. FAVRE, D. PORTAL, G. MAZARE, R. MORIN

Projet Socrate. Nouvelles spécifications (Version 3).

Université Scientifique et Médicale de Grenoble.
Grenoble, Septembre 1972.

[3] J.R. ABRIAL

Data Semantics.

Université Scientifique et Médicale de Grenoble.
Laboratoire d'Informatique. Novembre 1973.

[4] J.D. ARON

Information System in Perspective.

Computing Surveys Vol. 1, Décembre 1969.

[5] E. BENCI, C. CHARRET

L'analyse en question.

Dans Informatique et Gestion N° 45[23].

[6] S.C. BLUMENTHAL

Management information systems : a framework for planning and development.

Prentice Hall 1969.

[7] A. de CHELMINSKI

Macsil.

Thèse d'Ingénieur-Docteur.
Université Scientifique et Médicale de
Grenoble (à paraître Juin 1975).

[8] D. COMONT

Gestion des bibliothèques de programmes dans un grand projet.
Les techniques de l'informatique.
Congrès AFCET 1972.

[9] J.D. COUGER

Evaluation of Business System Analysis Techniques.
ACM Computing Surveys, Vol. 5 n° 3, Septembre 1973.

[10] J. COURTIN, J. VOIRON

Introduction à l'algorithmique et aux structures de données.
Cours polycopié - I.U.T.-B - Département Informatique.
Université des Sciences Sociales de Grenoble. 1974.

[11] CXP

Etude comparative des méthodes d'analyse.
Centre d'eXpérimentation des Packages - Etude 008.

[12] C. DELOBEL

Contributions théoriques à la conception et l'évaluation d'un système d'informations appliqué à la gestion.
Université Scientifique et Médicale de Grenoble.
Thèse d'Etat. Octobre 1973.

[13] J.C. DERNIAME

Un système de programmation modulaire : CIVA.
Thèse d'Etat - Nancy Janvier 1974.

[14] D. DUFAU

L'Audit d'application.
Informatique et gestion N° 53 - Décembre 1973.

[15] EDP Analyser

Managing the programming effort.
Juin 1968.

[16] EDP Analyser

Managing the systems effort.
Juillet 1968.

[17] EDP Analyser

The current status of data management.

Février 1974.

[18] J.C. FAVRE

Application SOCRATEP.

Projet Socrate - Note technique n° 23 Juin 1973.
Université Scientifique et Médicale de Grenoble.
Laboratoire d'Informatique.

[19] J.W. FORRESTER

Industrial Dynamics.

John Wiley 1961.

[20] GRADIA

Le projet MEDOC.

Note interne Mai 1975.
Groupe d'Appui pour le Développement de l'Informatique dans l'Administration - 27, Quai Anatole France PARIS (7e).

[21] F. GRUENBERGER

Critical factors in data management.

Prentice Hall 1969.

[22] J.J. HORNING et B. RANDELL

Process Structuring.

ACM Computing Surveys Vol. 5 N° 1 Mars 1973.

[23] INFORMATIQUE et GESTION N° 45

Dossier : les méthodes d'analyse.

(Sont succintement présentées les méthodes METALOG, MINOS, ARMIN, PARM, PROTEE, ARIANE).

[24] IRIA - SESORI

Proposition de recherche en Informatique appliquée aux organisations.

IRIA - Juin 1973.

[25] H. LESCA

Les ambiguïtés de l'analyse.

Dans Informatique et Gestion N° 45[23].

[26] J. MELESE

La gestion par les systèmes.

Editions Hommes et Techniques 1968.

[27] Ch. A. MYERS

The impact of Computers on Management.

The MIT Press 1967.

[28] F. NICK, C.P. OHLEN, V. SCHAEDEL, R. SCHAEFER

Systeme der computergestützen Systemgestaltung.

BIFOA Arbeits bericht 1972.

WISON VERLAG KOLN.

[29] J. ORLICKY

The successfull computer system.

Mc Graw Hill 1969.

[30] Y. ARCHER, C. DELOBEL, F. PECCOUD

Méthode générale d'analyse.

Rapport final Contrat DGRST 67 01 015 - Juillet 1969.

[31] F. PECCOUD

Syntactic and Semantic structures of files.

File 68 - Sponsored by IFIP administrative data processing group.

Copenhague. Novembre 1968.

[32] F. PECCOUD

Réflexions sur la construction d'un modèle d'aide à la conception des systèmes d'information.

Ecole d'Eté de l'AFCET - ALES Juillet 1971.

[33] F. PECCOUD et al.

MACSI : un modèle d'aide à la conception des systèmes informatiques.
Colloque "Les outils de l'analyse en informatique de gestion". Universités de Grenoble - Octobre 1971.

[34] F. PECCOUD

Ein Modell für die interacktive computergestützte Entwicklung grosser Informationssysteme.
BIFOA. Köln - Août 1973.

[35] J.C. ROCHETTE

Les méthodes d'analyse et de programmation.
CEGOS - Note interne.

[36] W.P. STEVENS, G.J. MYERS and L.L. CONSTANTINE

Structured design.
IBM Systems Journal. Vol. 13 N°2 1974.

[37] A. STIERS

Manuel utilisateur SOCRATE - Version 4.

[38] D. TEICHROEW et H. SAYANI

Automatisation de la construction de systèmes.
L'Informatique - DUNOD - Décembre 1971.

[39] D. TEICHROEW

A Survey of languages for stating requirements for computer-based information systems.
Fall Joint Computer Conference 1972.

[40] D. TEICHROEW et Al.

An Introduction to computer-aided documentation of user requirements for computer based information processing systems.
BIFOA Août 1973.

[41] D. TEICHROEW, E.A. HERSHEY, M.J. BASTARACHE

An Introduction to PSL/PSA.
ISDOS WORKING paper N° 86 - Mars 1974.
The University of Michigan - Ann Arbor Michigan 48104.

[42] TISSERAND et GUIARD

O.D.A.S.S. Outil de Documentation Automatique des Symboles d'un Système.

Congrès AFCET 1972.

[43] D.A. WALSH

A guide for software documentation.

Mc Graw Hill - Advanced Computer techniques Corporation - 1969.