HAL
open science

Macro-mécanisme pour le langage de commande d'un système conversationnel

Max Creveuil

To cite this version:

Max Creveuil. Macro-mécanisme pour le langage de commande d'un système conversationnel. Modélisation et simulation. Institut National Polytechnique de Grenoble - INPG; Université Joseph-Fourier - Grenoble I, 1974. Français. NNT : . tel-00284657

HAL Id: tel-00284657

https://theses.hal.science/tel-00284657

Submitted on 3 Jun 2008

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

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

THESE

présentée à

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

par

pour obtenir le grade de

Docteur de troisième cycle

Spécialité : INFORMATIQUE

Max CREVEUIL

MACRO – MECANISME POUR LE LANGAGE DE COMMANDE D'UN SYSTEME CONVERSATIONNEL

Thèse soutenue le 26 Juillet 1974 devant la Commission d'Examen :

Président	:	Monsieur L. BOLLIET
Examinateurs		Monsieur C. HANS
		Monsieur Y. SIRET
		Monsieur G. VEILLON

UNIVERSITE SCIENTIFIQUE
ET MEDICALE DE GRENOBLE

LISTE DES PROFESSEURS

Président : Monsieur Michel SOUTIF
Vice-Président : Monsieur Gabriel CAU

PROFESSEURS TITULAIRES

MM.	ANGLES D'AURIAC Paul	Mécanique des fluides
	ARNAUD Georges	Clinique des maladies infectieuses
	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
	BENOIT Jean	Radioélectricité
	BERNARD Alain	Mathématiques Pures
	BESSON Jean	Electrochimie
	BEZES Henri	Chirurgie générale
	BLAMBERT Maurice	Mathématiques Pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Georges	Electrotechnique
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET-EYMARD Joseph	Pathologie médicale
	BONNIER Etienne	Electrochimie Electrométallurgie
	BOUCHERLE André	Chimie et Toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BOUSSARD Jean-Claude	Mathématiques Appliquées
	BRAVARD Yves	Géographie
	BRISSONNEAU Pierre	Physique du solide
	BUYLE-BODIN Maurice	Electronique
	CABANAC Jean	Pathologie chirurgicale
	CABANEL Jean	Clinique rhumatologique et hydrologie
	CALAS François	Anatomie
	CARRAZ Gilbert	Biologie animale et pharmacodynamie
	CAU Gabriel	Médecine légale et Toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques Pures
	CHARACHON Robert	Oto-Rhino-Laryngologie
	CHATEAU Robert	Thérapeutique
	CHENE Marcel	Chimie papetière
	COEUR André	Pharmacie chimique
	CONTAMIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie Pathologique
	CRAYA Antoine	Mécanique

Mme	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBELMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DESRE Pierre	Métallurgie
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée
	DOLIQUE Jean-Michel	Physique des plasmas
	DREYFUS Bernard	Thermodynamique
	DUCROS Pierre	Cristallographie
	DUGOIS Pierre	Clinique de Dermatologie et Syphiligraphie
	FAU René	Clinique neuro-psychiatrique
	FELICI Noël	Electrostatique
	GAGNAIRE Didier	Chimie physique
	GALLISSOT François	Mathématiques Pures
	GALVANI Octave	Mathématiques Pures
	GASTINEL Noël	Analyse numérique
	GEINDRE Michel	Electroradiologie
	GERBER Robert	Mathématiques Pures
	GIRAUD Pierre	Géologie
	KLEIN Joseph	Mathématiques Pures
Mme	KOFLER Lucie	Botanique et Physilogie végétale
MM.	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
	LOUP Jean	Géographie
Mle	LUTZ Elisabeth	Mathématiques Pures
MM.	MALGRANGE Bernard	Mathématiques Pures
	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Seméiologie médicale
	MASSEPORT Jean	Géographie
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et Pétrographie
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	NEEL Louis	Physique du solide
	OZENDA Paul	Botanique
	PAUTHENET René	Electrotechnique
	PAYAN Jean-Jacques	Mathématiques Pures
	PEBAY-PEYROULA Jean-Claude	Physique
	PERRET René	Servoméçanismes
	PILLET Emile	Physique industrielle
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	REULOS René	Physique industrielle
	RINALDI Renaud	Physique
	ROGET Jean	Clinique de pédiatrie et de puériculture
	SANTON Lucien	Mécanique
	SEIGNEURIN Raymond	Microbiologie et Hygiène
	SENGEL Philippe	Zoologie
	SILBERT Robert	Mécanique des fluides
	SOUTIF Michel	Physique générale

MM.	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLAND François	Zoologie
	VALENTIN Jacques	Physique nucléaire
	VAUQUOIS Bernard	Calcul électronique
Mme	VERAIN Alice	Pharmacie galénique
M.	VERAIN André	Physique
Mme	VEYRET Germaine	Géographie
MM.	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale
	YOCCOZ Jean	Physique nucléaire théorique

PROFESSEURS ASSOCIES

MM.	BULLEMER Bernhard	Physique
	HANO JUN-ICHI	Mathématiques Pures
	STEPHENS Michaël	Mathématiques appliquées

PROFESSEURS SANS CHAIRE

MM.	BEAUDOING André	Pédiatrie
Mme	BERTRANDIAS Françoise	Mathématiques Pures
MM.	BERTRANDIAS Jean-Paul	Mathématiques appliquées
	BIAREZ Jean-Pierre	Mécanique
	BONNETAIN Lucien	Chimie minérale
Mme	BONNIER Jane	Chimie générale
MM.	CARLIER Georges	Biologie végétale
	COHEN Joseph	Electrotechnique
	COUMES André	Radioélectricité
	DEPASSEL Roger	Mécanique des fluides
	DEPORTES Charles	Chimie minérale
	GAUTHIER Yves	Sciences biologiques
	GAVEND Michel	Pharmacologie
	GERMAIN Jean-Pierre	Mécanique
	GIDON Paul	Géologie et Minéralogie
	GLENAT René	Chimie organique
	HACQUES Gérard	Calcul numérique
	JANIN Bernard	Géographie
Mme	KAHANE Josette	Physique
MM.	MULLER Jean-Michel	Thérapeutique
	PERRIAUX Jean-Jacques	Géologie et Minéralogie
	POULOUJADOFF Michel	Electrotechnique
	REBECQ Jacques	Biologie (CUS)
	REVOL Michel	Urologie
	REYMOND Jean-Charles	Chirurgie générale
	ROBERT André	Chimie papetière
	DE ROUGEMONT Jacques	Neurochirurgie
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIBILLE Robert	Construction mécanique
	SIROT Louis	Chirurgie générale
Mme	SOUTIF Jeanne	Physique générale

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

Mle	AGNIUS-DELORD Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	AMBLARD Pierre	Dermatologie
	AMBROISE-THOMAS Pierre	Parasitologie
	ARMAND Yves	Chimie
	BEGUIN Claude	Chimie organique
	BELORIZKY Elie	Physique
	BENZAKEN Claude	Mathématiques appliquées
	BILLET Jean	Géographie
	BLIMAN Samuel	Electronique (EIE)
	BLOCH Daniel	Electrotechnique
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BOUCHET Yves	Anatomie
	BOUVARD Maurice	Mécanique des fluides
	BRODEAU François	Mathématiques (IUT B)
	BRUGEL Lucien	Energétique
	BUISSON Roger	Physique
	BUTEL Jean	Orthopédie
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse
	CHIAVERINA Jean	Biologie appliquée (EFP)
	CHIBON Pierre	Biologie animale
	COHEN-ADDAD Jean-Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie médicale
	CONTE René	Physique
	COULOMB Max	Radiologie
	CROUZET Guy	Radiologie
	DURAND Francis	Métallurgie
	DUSSAUD René	Mathématiques (CUS)
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	GENSAC Pierre	Botanique
	GIDON Maurice	Géologie
	GRIFFITHS Michaël	Mathématiques appliquées
	GROULADE Joseph	Biochimie médicale
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Hygiène et Médecine préventive
	IDELMAN Simon	Physiologie animale
	IVANES Marcel	Electricité
	JALBERT Pierre	Histologie
	JOLY Jean-René	Mathématiques Pures
	JOUBERT Jean-Claude	Physique du solide
	JULLIEN Pierre	Mathématiques Pures
	KAHANE André	Physique générale
	KUHN Gérard	Physique
	LACOUME Jean-Louis	Physique
Mme	LAJZEROWICZ Jeannine	Physique
MM.	LANCIA Roland	Physique atomique
	LE JUNTER Noël	Electronique
	LEROY Philippe	Mathématiques
	LOISEAUX Jean-Marie	Physique nucléaire
	LONGEQUEUE Jean-Pierre	Physique nucléaire
	LUU DUC Cuong	Chimie organique
	MACHE Régis	Physiologie végétale
	MAGNIN Robert	Hygiène et Médecine préventive
	MARECHAL Jean	Mécanique
	MARTIN-BOUYER Michel	Chimie (CUS)

MM.	MAYNARD Roger	Physique du solide
	MICHOULIER Jean	Physique (IUT A)
	MICOUD Max	Maladies infectieuses
	MOREAU René	Hydraulique (INP)
	NEGRE Robert	Mécanique
	PARAMELLE Bernard	Pneumologie
	PECCOUD François	Analyse (IUT B)
	PEFFEN René	Métallurgie
	PELMONT Jean	Physiologie animale
	PERRET Jean	Neurologie
	PERRIN Louis	Pathologie expérimentale
	PFISTER Jean-Claude	Physique du solide
	PHELIP Xavier	Rhumatologie
Mlle	RIERY Yvette	Biologie animale
MM.	RACHAIL Michel	Médecine interne
	RACINET Claude	Gynécologie et obstétrique
	RENAUD Maurice	Chimie
	RICHARD Lucien	Botanique
Mme	RINAUDO Marquerite	Chimie macromoléculaire
MM.	ROMIER Guy	Mathématiques (IUT B)
	SHOM Jean-Claude	Chimie générale
	STIEGLITZ Paul	Anesthésiologie
	STOEBNER Pierre	Anatomie pathologique
	VAN CUTSEM Bernard	Mathématiques appliquées
	VEILLON Gérard	Mathématiques appliquées (INP)
	VIALON Pierre	Géologie
	VOOG Robert	Médecine interne
	VROUSSOS Constantin	Radiologie
	ZADWORNY François	Electronique

MAITRES DE CONFERENCES ASSOCIES

MM.	BOUDOURIS Georges	Radioélectricité
	CHEEKE John	Thermodynamique
	GOLDSCHMIDT Hubert	Mathématiques
	SIDNEY STUARD	Mathématiques Pures
	YACOUD Mahmoud	Médecine légale

CHARGES DE FONCTIONS DE MAITRES DE CONFERENCES

Mme	BERIEL Hélène	Physilogie
Mme	RENAUDET Jacqueline	Microbiologie

Fait le 30 mai 1972.

M A C R O - M E C A N I S M E D ' E X T E N S I O N

P O U R

L E L A N G A G E D E C O M M A N D E

D ' U N

S Y S T E M E C O N V E R S A T I O N N E L

Je souhaite remercier :

Monsieur le professeur Louis BOLLIET, qui a bien voulu me faire l'honneur de présider le jury de cette thèse.

Messieurs Gérard VEILLON et Yvon SIRET, qui se sont intéressés à mon travail et ont accepté de le juger.

Monsieur Claude HANS, du Centre Scientifique IBM de Grenoble, qui est à l'origine de ce travail et auprès de qui j'ai pu trouver expérience, conseils et temps qui m'ont été précieux pour me diriger et m'encourager dans ce travail.

Tous mes collègues du Centre Scientifique pour leur aide et leurs critiques constructives, en particulier Xavier de LAMBERTERIE et Jorge VIDART.

J'exprime ma reconnaissance à la compagnie IBM-FRANCE grâce à qui j'ai pu réaliser ce travail.

Je ne saurais oublier le Service de Reproduction pour la réalisation matérielle de cet ouvrage.

RESUME DE LA THESE

Le travail réalisé comprend une réflexion sur la structure de base des langages de commande puis la proposition (avec réalision effective sous forme d'un prototype opérationnel) d'un macro-mécanisme, enrichi par rapport à ceux existants, qui permet d'accroître la puissance d'un langage de commande en lui fournissant des mécanismes d'extension.

L'étude critique des mécanismes de langages de commande évolués, plus particulièrement celui de CMS/370, a permis d'isoler le concept d'environnement ainsi que ses divers états possibles. Grâce à l'existence sous-jacente de telles structures, le langage de commande est modulaire et potentiellement extensible (ce qui permet, en particulier, de le personnaliser au gré des besoins des divers classes d'utilisateurs). En exploitant plus à fond ces propriétés, il est alors envisageable de réaliser un langage de commande primitif très modulaire qui est ensuite étendu grâce à un puissant macro-mécanisme. C'est cette approche qui a donné lieu à la réalisation du prototype EXEC+ (construit sous CMS) décrit dans la seconde partie du mémoire.

TABLE DES MATIERES

INTRODUCTION..... i1

C H A P I T R E I

STRUCTURE ET CONTROLE DES COMMANDES..... 1.1

1 STRUCTURE D'UN LANGAGE DE COMMANDE..... 1.2

1.1 COMMANDE DE BASE..... 1.2

1.2 NOTION D'ENVIRONNEMENT..... 1.4

1.3 ETAT D'UN ENVIRONNEMENT..... 1.7

1.4 COMMUNICATION ENTRE ENVIRONNEMENT..... 1.13

2 CONTROLE DE L'EXECUTION DES COMMANDES..... 1.19

2.1 CONCEPT DE MACRO..... 1.21

2.1.1 MACRO-ASSEMBLEUR..... 1.22

2.1.2 MACRO-PROCESSEUR..... 1.23

2.1.3 MACRO-INTERPRETEUR..... 1.24

2.2 CONSIDERATIONS SYNTAXIQUE..... 1.25

2.2.1 SYNTAXE DE L'APPEL..... 1.25

2.2.2 SYNTAXE DES PARAMETRES FORMELS..... 1.27

2.2.3 SYNTAXE DU LANGAGE DE CONTROLE..... 1.27

2.3 FACILITES DU MACRO-LANGAGE..... 1.30

2.3.1 CHAINE DE CARACTERES..... 1.31

2.3.2 VARIABLES INTERNES..... 1.33

2.3.3 INSTRUCTIONS DE CONTROLE..... 1.34

C H A P I T R E I I

DESCRIPTION EXTERNE DE EXEC+..... 2.1

1 CREATION D'UNE MACRO-COMMANDE EXEC..... 2.2

2 APPEL D'UNE MACRO-COMMANDE..... 2.3

3 CARACTERISTIQUES GENERALES DES MACRO-COMMANDES..... 2.4

3.1 LIGNES D'UNE MACRO-COMMANDE..... 2.4

3.1.1 COMMANDES CMS..... 2.4

3.1.2 LANGAGE DE CONTROLE EXEC..... 2.5

3.2 VARIABLES..... 2.6

3.3 CONCATENATION..... 2.8

4 MANUEL DE L'UTILISATEUR DE EXEC+ SOUS CMS..... 2.9

4.1 MOT - DELIMITEUR..... 2.9

4.2 CHAINE DE CARACTERES..... 2.9

4.3 SUBSTITUTION..... 2.10

4.4 DESCRIPTION DE LA COMMANDE EXEC..... 2.13

4.5 DESCRIPTION DES REQUETES DU LANGAGE DE CONTROLE..... 2.16

&ARGS..... 2.16

&BEGPUNCH..... 2.18

&BEGSTACK..... 2.20

&BEGTYPE..... 2.22

&CONTINUE..... 2.24

&CONTROL..... 2.25

&END..... 2.27

&ERROR..... 2.28

&EXIT..... 2.30

&GOTO..... 2.31

&IF..... 2.32

&LOOP..... 2.34

&PUNCH..... 2.36

&READ..... 2.37

&SKIP..... 2.39

&SPACE..... 2.40

&STACK..... 2.41

&TIME..... 2.43

&TYPE..... 2.45

5 FONCTIONS INTERNES..... 2.46

5.1 &DATATYPE..... 2.46

5.2 &LENGTH..... 2.47

5.3 &SUBSTR..... 2.47

6 VARIABLES SPECIALES..... 2.48

6.1 VARIABLES SUR L'ETAT DE LA MACRO-COMMANDE..... 2.48

6.2 VARIABLES SUR L'ETAT DE LA MACHINE VIRTUELLE..... 2.49

6.3 VARIABLES GENERALES..... 2.51

7 DESCRIPTION DES MESSAGES D'ERREUR..... 2.51

C H A P I T R E I I I

DESCRIPTION INTERNE ET LOGIQUE DE EXEC+..... 3.1

1 PRESENTATION..... 3.2

2 DESCRIPTION INTERNE DE DMSEXC..... 3.3

3 L'INTERPRETEUR DE COMMANDES..... 3.4

3.1 UTILISATION DU LANGAGE D'ECRITURE..... 3.4

3.2 EXECTOR: ANALYSE D'UNE MACRO-COMMANDE..... 3.6

3.2.1 GESTION GLOBALE DES VARIABLES DE L'INTERPRETEUR..... 3.6

3.2.2 CHOIX DE LA COMMANDE..... 3.9

3.2.3 ANALYSE ET SUBSTITUTION DANS LA COMMANDE..... 3.12

3.3 SUBSTIT: MECANISME DE SUBSTITUTION..... 3.15

3.4 STARTCOM: INTERPRETATION DES COMMANDES..... 3.18

3.4.1 INTERPRETATION DES REQUETES DE CONTROLE..... 3.19

3.4.1.1 TRAITEMENT DES PRINCIPALES COMMANDES.. 3.19

3.4.1.2 DESCRIPTION INTERNE DES AFFECTATIONS.. 3.21

3.4.2 INTERPRETATION DES COMMANDES..... 3.26

CONCLUSION..... c.1

BIBLIOGRAPHIE..... B.1

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

Pour communiquer avec un système d'exploitation, que ce soit un système "batch" ou un système conversationnel, l'utilisateur a besoin de passer par un interface qui est le langage de commande du système. L'expérience semble montrer que ces langages de commande font partie d'un domaine un peu oublié parmi les travaux de recherches. Ceci provient peut-être de la difficulté à les situer. Pour ceux qui concoivent les systèmes, définir le langage de commande n'est sans doute pas la principale préoccupation, en ce sens que des objectifs du système découleront les commandes. Et pour ceux qui concoivent les langages ceux-ci sont à la fois trop simples et trop dépendants des systèmes pour appartenir à leur domaine. C'est à ce problème des langages de commande que nous nous sommes intéréssé.

Le langage de commande en tant qu'interface de communication logique entre l'utilisateur et le système, ne peut répondre totalement aux besoins de tout utilisateur. Ceci provient de ce qu'il est techniquement impossible et peut-être même pas souhaitable de fournir un interface suffisamment vaste pour satisfaire les besoins de tous les utilisateurs. La solution qui peut être envisagée pour résoudre ce problème est non plus de fournir un système de communication figé, mais plutôt la possibilité pour chacun de se définir un langage de commande "sur mesure".

Les avantages que présente cette philosophie générale au niveau de la conception globale d'un système sont multiples. Sans vouloir être exhaustif, nous en citerons quelques uns

Premièrement, nous pouvons constater que les mêmes séquences de commandes sont plusieurs fois exécutées par un même utilisateur (réf. GRANT). Par exemple il pourra avoir très souvent une phase compilation, impression et exécution d'un programme. Il est alors intéressant dans ce cas de pouvoir regrouper ces commandes en une seule. L'intérêt que cela présente est double. D'une part cela permet à l'utilisateur de s'exprimer dans une forme beaucoup plus condensée. D'où une plus grande facilité d'emploi et un risque moindre de fausses manipulations dû à la restriction d'information qu'a à fournir l'utilisateur. D'autre part, l'enchaînement automatique des commandes représente un gain de temps important puisqu'il n'y a plus d'intervention manuelle, d'où une productivité accrue. Nous pouvons citer à ce propos les mesures faites par W.J. DOHERTY sur le système TSS/360 d'IBM au centre de recherche T.J. WATSON de New York (réf. DOHERTY). Il se propose d'analyser les conséquences de l'emploi des systèmes "batch" ou conversationnels sur le plan de la productivité des utilisateurs. S'il a mis en évidence l'amélioration de cette productivité lors du passage d'un système "batch" à un système conversationnel, il a surtout montré le phénomène

d'adaption de l'utilisateur à un système conversationnel qui fait implicitement ressortir la notion de connaissance de plus en plus complète du système qui peut induire une charge de plus en plus lourde. Charge qui sera d'autant plus importante d'ailleurs que le système fournira des outils d'expansion de commandes. C'est ainsi qu'il a pu mesurer que certains utilisateurs avaient, individuellement et de façon répétée, demandé l'exécution de 22000 commandes de base en moyenne par jour; alors que la somme des commandes exécutées par 90 autres utilisateurs n'était que de 20000. Il explique cette différence, en plus du phénomène d'adaption mentionné, par l'utilisation intensive que faisait le premier groupe d'utilisateur du mécanisme d'expansion de commandes.

Le second avantage présenté par cette idée de permettre à l'utilisateur de se définir un langage de commande "sur mesure" est ce que nous appellerons la personnalisation de l'interface de communication. C'est à dire pouvoir adapter à chaque utilisation particulière un ensemble de commandes mieux appropriées (réf. SAVARY). Ce point nous semble important dans la mesure où les buts des divers utilisateurs sont très variés. Et il est difficilement pensable avec un système de communication unique, de satisfaire également l'ensemble des utilisateurs (réf. DOLOTTA). Nous pensons que la possibilité offerte permet d'apporter une plus grande souplesse d'utilisation des systèmes.

Enfin le dernier point que nous mentionnerons est relatif à la conception globale du système. En effet, en permettant à chaque utilisateur de définir son propre système de communication, il résulte que l'interface qui reste à la charge du concepteur se restreint. L'idée générale à ce niveau est alors de créer un interface de base que chacun pourra étendre selon ses propres besoins. Cela revient à réduire l'ensemble des commandes de bases à celles qui définissent des actions élémentaires (réf. HANS). Ce que nous pouvons exprimer par la notion de conception par couche, dans la quelle nous trouvons (dans l'ordre) : le hardware, le système, le langage de base, le langage étendu et l'utilisateur. Ceci peut à priori diminuer le volume global du système; mais par contre il exige un soin attentif de la part du concepteur dans la détermination des commandes de bases.

On s'aperçoit donc que l'idée générale, en ce qui concerne l'interface de communication dans un système qui consiste à donner le moyen à chaque utilisateur d'étendre un langage de base plutôt que de fournir un langage de commande suffisamment étendu pour satisfaire le plus grand nombre d'entre eux, présente un certains nombre de points intéressants. Cependant l'existence d'un tel mécanisme impose des contraintes dans la conception du système qui doit en conséquence fournir certaines possibilités et satisfaire certains critères.

Ainsi par exemple, on doit trouver la possibilité de demander l'exécution d'une commande non seulement à partir d'une console, s'il s'agit d'un système conversationnel, ou de cartes (par exemple), s'il s'agit d'un système "batch" mais aussi à partir de n'importe quel programme utilisateur. C'est ce que DOLOTTA et IRVINE appellent "le chargement et l'exécution implicite" (réf. DOLOTTA). Ceci traduit la possibilité de demander, à l'intérieur d'un programme utilisateur, que soit chargé le module qui interprètera telle commande et que le contrôle lui soit passé. Si cette possibilité n'est pas offerte, il est alors impossible pour un utilisateur d'écrire un mécanisme d'expansion qui ne soit pas intégré au système. Le but d'un tel mécanisme est en effet, outre la partie acquisition de commandes à partir d'un fichier ou d'une procédure cataloguée par exemple, de faire exécuter des commandes. Ceci ne pourra se faire que s'il existe la possibilité pour tout programme utilisateur de demander l'exécution de commandes de base.

Le fait que chaque utilisateur puisse étendre le langage de base peut amener ce que nous appellerons des conflits de nom. Nous disons qu'il y a conflit de nom lorsqu'une commande du langage étendu définie par l'utilisateur, a même nom qu'une commande du langage de base. C'est par exemple le cas lorsque l'utilisateur se définit une commande du langage étendu de nom ASSEMBLE et

qu'il existe une commande de base de même nom. Lors de la conception du système il est donc nécessaire de prévoir la résolution de ces cas afin de pouvoir résoudre les ambigüités possibles.

Nous voyons donc qu'un mécanisme d'extension pour un langage de commande n'est envisageable que si le système répond à certains critères.

Dans le premier chapitre nous faisons donc, dans une première partie, une étude critique des mécanismes de langages de commande évolués. Celle-ci porte en particulier sur le langage de commande de CMS/370 dont l'étude a posteriori a permis de mettre en évidence des concepts sous-jacents, tel celui d'environnement, qui le rendent potentiellement extensible.

Dans une seconde partie, nous essayons de situer le mécanisme d'extension pour un langage de commande par rapport à la notion de macro-processeur telle que l'a définie P.J. BROWN. Nous devons en effet constater que si de nombreuses études ont été faites dans ce domaine, elles sont relatives uniquement aux mécanismes d'extension pour les langages assembleurs ou les langages de haut niveau (réf. BROWN). Ceci provient de ce que les langages de commandes n'ont pas fait l'objet d'études aussi approfondies que pour les autres langages. C'est dû au fait que leur étude n'est

plus tout à fait du domaine des systèmes et pas encore du domaine des langages. Nous serons ensuite conduit a avoir quelques considérations vis à vis d'un langage de contrôle d'une part sur un plan syntaxique, d'autre part sur les facilités offertes.

Dans le second chapitre, nous faisons une description externe du mécanisme d'extension que nous proposons pour CMS/370 et dont nous avons réalisé un prototype (EXEC+).

Le troisième chapitre enfin, présente la structuration interne du mécanisme proposé et décrit l'implantation qui en a été faite.

C H A P I T R E I

S T R U C T U R E E T C O N T R O L E
D E S C O M M A N D E S

1 STRUCTURE D'UN LANGAGE DE COMMANDE

Nous faisons dans cette première partie une étude critique de la structure d'un langage de commande évolué et de ses mécanismes. Le but de celle-ci est de mettre en évidence un certains nombre de concepts qui permettent d'envisager alors un mécanisme d'extension du langage.

Cette étude porte plus particulièrement sur le système conversationnel CMS/370. L'analyse a posteriori de ce système nous a permis de dégager des notions comme celle d'environnement et de définir des états pour ceux-ci ainsi que d'expliciter les mécanismes de communication entre eux. Ceci nous permettra de donner une description dynamique du système.

1.1 COMMANDE DE BASE

Le langage de commande considéré est celui d'un système interactif. C'est à dire qu'à tout moment l'utilisateur peut adresser des requêtes au système. L'interface de communication physique entre l'utilisateur et le système est la console. C'est par son intermédiaire que l'utilisateur peut adresser ses requêtes. Sur un plan logique, le moyen de communiquer avec le système est l'utilisation d'un langage de commande. Ce langage est constitué d'un ensemble de

commandes de bases. Cet ensemble est défini par l'interpréteur du langage. Une commande de base est définie comme une ligne logique. Elle est elle-même un ensemble composé de mots. Par définition le premier mot de la ligne sera appelé le nom de la commande; les autres mots constitueront une liste de paramètres qui lui est associée. Les paramètres rencontrés peuvent être de plusieurs types. Ainsi nous pouvons trouver une donnée littérale comme par exemple dans la commande :

MSG OPERATEUR ATTACHER LA BANDE 135

où les mots ATTACHER LA BANDE 135 constituent une donnée littérale. Le mot OPERATEUR est quant à lui un objet référencé par nom. Nous pouvons également trouver une référence par nom à une collection d'objets. Par exemple dans la commande

PRINT TAUX FORTRAN

le nom TAUX FORTRAN fait référence à une collection d'objets qui constituent un fichier.

En plus de cette différenciation au niveau des paramètres dans une commande, nous en trouvons une autre au niveau de l'interprétation des commandes. Nous pouvons en effet constater que l'interpréteur de certaines commandes est un programme interactif alors que pour d'autres commandes il ne l'est pas. Ainsi par exemple dans le cas de la commande PRINT ci-dessus, entre le moment où l'interpréteur prend le contrôle et celui où il le rend à

l'utilisateur, aucune communication ne s'est faite entre le système (c'est à dire l'interpréteur) et l'utilisateur. Par contre dans le cas de la commande EDIT par exemple, le déroulement de l'interprétation de cette commande est différent. A partir du moment où l'interpréteur prend le contrôle, il y a à nouveau interaction entre celui-ci et l'utilisateur qui peut adresser de nouvelles requêtes.

C'est cette classification des commandes de base du langage de commande qui va nous permettre de définir une structure telle, que ce langage de commande ne sera pas perçu comme un tout, mais au contraire comme composé d'éléments disjoints avec cependant existence de connexions entre ces éléments.

1.2 NOTION D'ENVIRONNEMENT

Ainsi que nous l'avons déjà mentionné précédemment, le langage de commandes est défini par un interpréteur. C'est à dire qu'à un instant donné, l'ensemble des commandes utilisables est défini par l'interpréteur actif (qui a le contrôle). Donc lorsque nous avons dit que l'interprétation de certaines commandes de base est interactive, cela est équivalent à dire qu'il y a eu changement de l'interpréteur. Considérons par exemple le cas de la commande EDIT. Avant l'exécution de cette commande, l'utilisateur se trouve dans

un certain contexte : il peut utiliser un certain nombre de commandes (ASSEMBLE, FORTRAN etc...) qui portent sur un ensemble de données bien défini. Cet ensemble comprend entre autre l'ensemble des fichiers que possède l'utilisateur. Au moment de l'exécution de la commande EDIT, le langage de commande à la disposition de l'utilisateur a changé. Nous pouvons d'ailleurs noter qu'on parle dans le cas d'EDIT, de langage de requêtes. Ce langage comprend par exemple des commandes d'insertion, d'effacement ou de modification de lignes dans un fichier dont le nom est apparu en paramètre dans la commande EDIT. D'autre part l'ensemble des données manipulables par ce langage de requêtes est constitué par les lignes du fichier. Donc, deux ensembles on été modifiés à l'exécution de cette commande; ce sont :

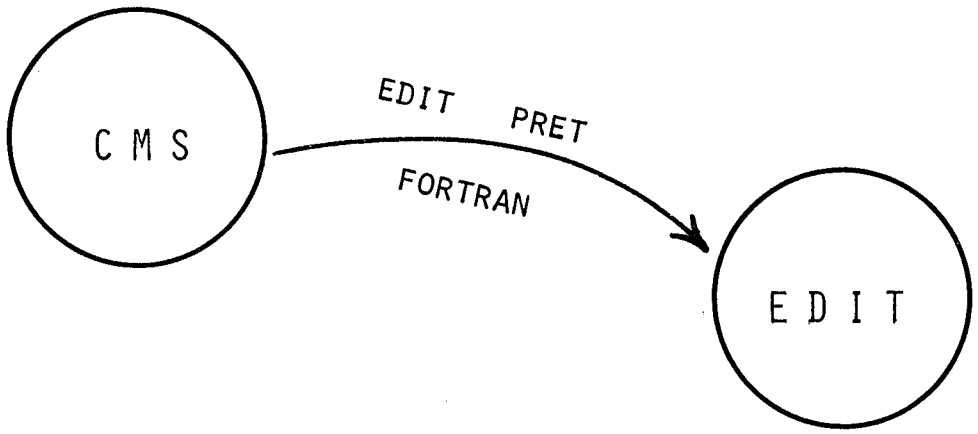
- le langage de commande.
- l'ensemble des données manipulables par ce langage.

D'où, la notion d'environnement :

Un environnement est, par définition, l'ensemble composé d'un langage de commande et des données manipulées par ce langage.

Il y a changement d'environnement lorsque l'interprétation d'une commande de base dans un langage donné est interactive. Ainsi, la figure 1 représente le passage de l'environnement CMS dans l'environnement EDIT après

exécution de la commande de base EDIT PRET FORTRAN.



CHANGEMENT D'ENVIRONNEMENT

Figure 1

Nous pouvons noter que par convention, nous donnons à un environnement le même nom que celui de la commande qui a provoqué le transfert. Dans le langage de commande d'un environnement nous trouvons donc deux types de commandes de base. Il y a d'une part les commandes dites internes, c'est à dire qui laissent l'utilisateur dans le même environnement. D'autre part il y a les commandes dites de transfert qui font passer dans un autre environnement.

Nous pouvons alors donner une représentation de la structure du langage de commandes en tenant compte des notions introduites précédemment. Les noeuds de la structure représentent les environnements et les connexions sont la

représentation des commandes de transfert pour chaque environnement.

Avec une telle structure le problème de l'extension du langage de commande ne se pose donc plus au niveau globale du langage de commandes, mais au niveau de chaque environnement. Nous pouvons en effet envisager l'extension du langage de commande de chaque environnement indépendamment des autres; ou avoir un mécanisme général utilisable à partir de chaque environnement. Nous pouvons pour cela envisager les relations qui peuvent exister entre les divers environnements à un instant donné.

1.3 ETAT D'UN ENVIRONNEMENT

Les connexions que nous avons représentées entre deux environnements à la figure 2, correspondent en fait à une analyse statique de la structure du système. Elles traduisent une représentation simplement hiérarchique de la structure qui ne fait pas intervenir la notion de temps. Or la connexion entre deux environnements n'est établie qu'à un instant donné : celui où il y a exécution d'une commande de transfert propre à l'environnement "appelant". Cette considération nous conduit à définir la notion d'état d'un environnement.

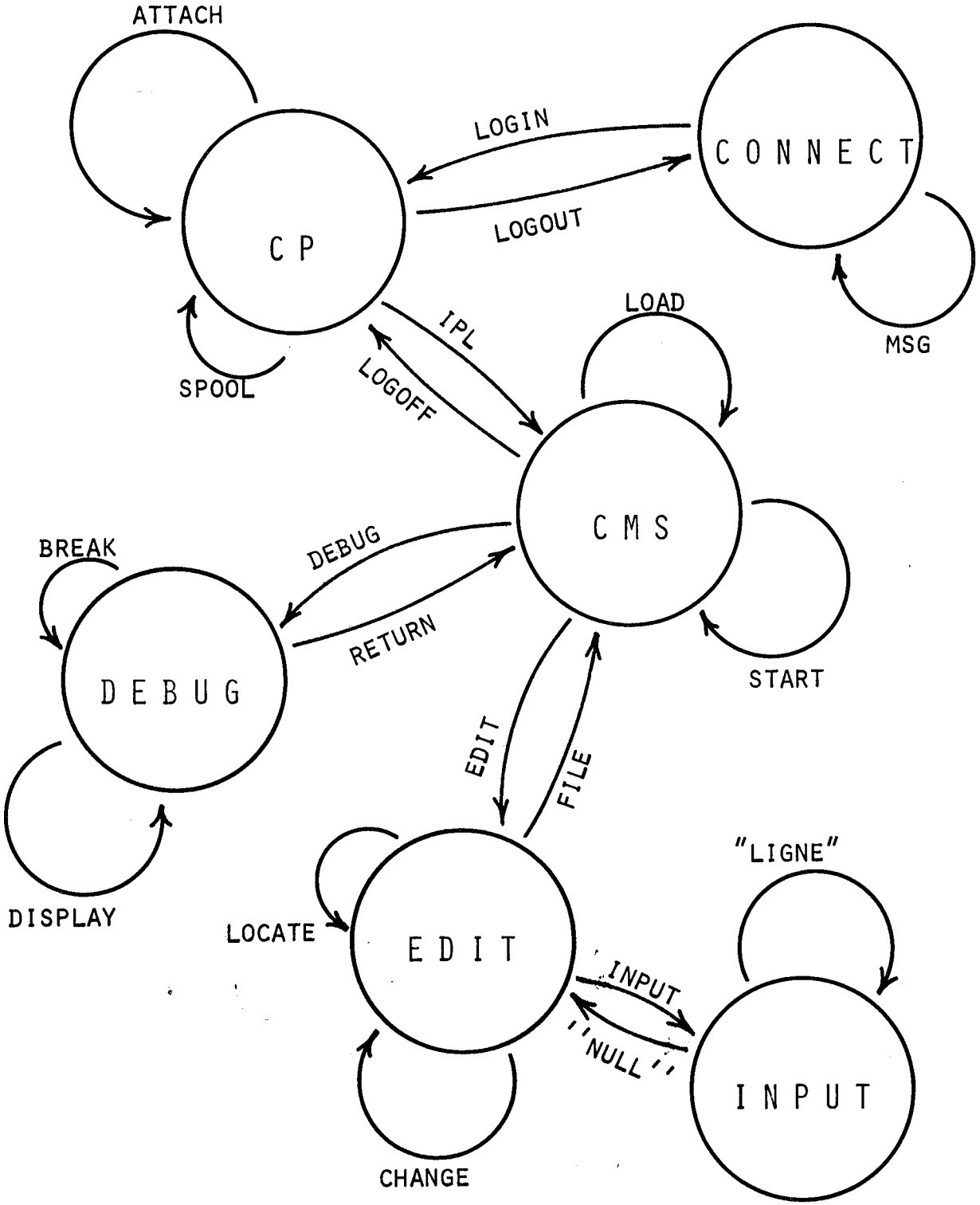


FIGURE 2

DESCRIPTION STATIQUE DE LA STRUCTURE

Lorsque la commande exécutée est une commande de transfert, l'environnement à partir duquel elle a été lancée passe dans l'état dormant. Quant au nouvel environnement, celui qui est "appelé", il est dit être dans l'état actif. Cet état se caractérise par le fait qu'à l'instant considéré c'est l'interpréteur de cet environnement qui a le contrôle. Enfin, dans tous les autres cas, un environnement se trouve être dans l'état non-actif.

En résumé nous avons trois états possibles pour un environnement :

- l'état ACTIF.
- l'état DORMANT.
- l'état NON-ACTIF.

La figure 3 donne une représentation temporelle de la structure. Vis à vis de l'utilisateur, la différence entre un environnement dormant et un environnement non-actif, provient de ce que l'interpréteur du premier a ce que nous appellerons une activité potentielle. L'utilisateur a en effet la possibilité de "réveiller" momentanément cet environnement à partir de l'environnement actif. C'est à dire qu'il peut demander l'exécution de commandes de base appartenant à un environnement dormant. Considérons la séquence de commandes suivante :

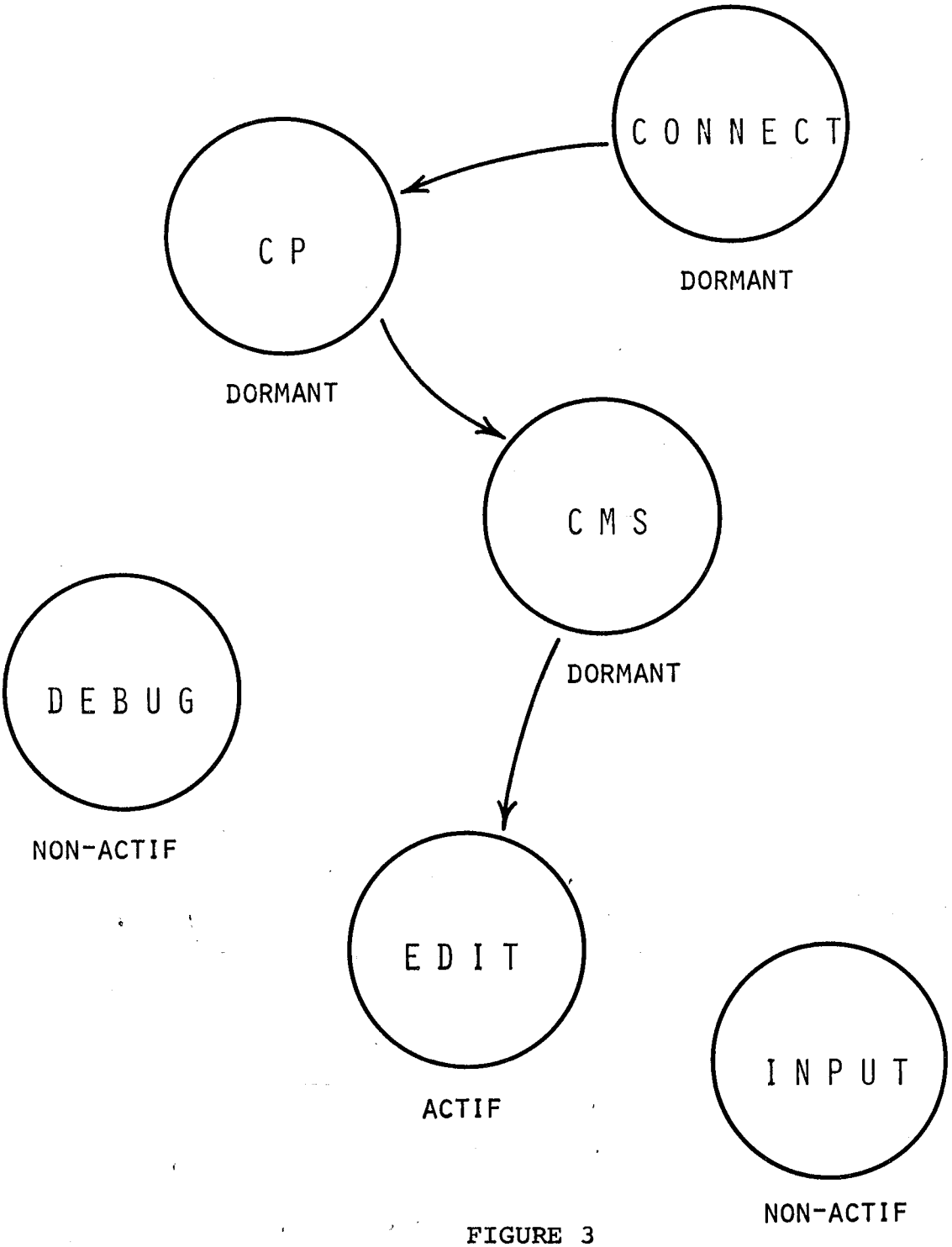


FIGURE 3

DESCRIPTION DYNAMIQUE DE LA STRUCTURE

Commande	Environnement actif
	CONNECT
LOGIN AMX	CP
IPL CMS	CMS
EDIT PRET FORTRAN	EDIT

Après exécution de la dernière commande EDIT, l'environnement actif est l'environnement EDIT. Bien que l'environnement CMS soit dormant l'utilisateur a néanmoins la possibilité, à partir d'EDIT, de lancer l'exécution de commandes de base de l'environnement CMS, puis de revenir sous EDIT dans le même état que lorsqu'il l'avait quitté. Cette possibilité ne lui est pas offerte pour les commandes de base de l'environnement DEBUG par exemple qui lui, se trouve dans l'état non-actif. C'est cette différence qui nous permet de dire qu'un environnement dormant a une "activité potentielle".

Cette notion d'état dormant est importante, dans l'optique d'une extension du langage, dans le cas de l'environnement CP. Comme nous l'avons dit précédemment, la fonction du système CP est de simuler une machine réelle pour plusieurs personnes en même temps. Une première conséquence, est que nous ne trouvons pas dans CP une gestion de fichiers pour les utilisateurs puisque cette notion est inconnue d'une machine réelle. Or, comme nous le verrons, le macro-mécanisme d'extension que nous proposons

utilise cette notion de fichier utilisateur, puisqu'une macro-commande sera définie comme un fichier particulier d'un certain type. Nous sommes donc en présence a priori d'un obstacle important pour pouvoir étendre le langage de commande propre à CP. D'après ce qui précède, une solution pour contourner cet obstacle consistera pour l'utilisateur à rendre l'environnement CMS actif, puis à partir de là, procéder par "réveils" successifs pour l'environnement CP. Ainsi, nous aurons une macro-commande CMS, dont les commandes de base qui seront générées appartiendront au langage de commande de CP. Cette macro-commande sera donc l'équivalent fonctionnel d'une macro-commande CP. Illustrons cette notion par un exemple.

Supposons que l'utilisateur veuille créer une macro-commande qui lui permette d'obtenir le résultat de ses travaux sur un listing selon une certaine présentation. Il nous faut rappeler que les sorties imprimantes des utilisateurs dans le système CP, transitent par un mécanisme particulier désigné sous le terme de "spooling" (réf. IBM CP). La macro-commande contiendra donc des commandes CP qui agissent sur ce mécanisme. La macro-commande qu'il créera pourra être la suivante :

```
nom de macro        IMPRIME
corps de macro      CP SPOOL PRINTER NOHOLD
                    CP CHANGE PRINTER DIST &1
```

CP CLOSE PRINTER

L'appel à cette macro-commande se fera à partir de l'environnement CMS en lançant sur la console la commande

IMPRIME MONFICHIER

Les commandes de bases qui seront générées sont toutes préfixées par le mot CP, ce qui aura pour effet de "réveiller" à chaque fois l'environnement CP. Sur un plan purement fonctionnel on a donc de cette façon une extension du langage de commande de l'environnement CP.

Avec cet exemple nous avons pu constater que l'indépendance entre les environnements n'est pas totale et que certaines communications peuvent exister entre eux.

1.4 COMMUNICATION ENTRE ENVIRONNEMENTS

Les communications entre deux environnements proviennent du fait qu'il est possible, à partir d'un environnement donné, de demander l'exécution de commandes appartenant au langage de commande d'un autre environnement. Cette possibilité nous importe dans la mesure où, pour certains environnements comme CP, la structure de l'interpréteur du langage de commande empêche l'utilisation du mécanisme d'extension du langage de base que nous proposons. Et, comme nous l'avons vu dans l'exemple

précédent, cette possibilité de communication permet de résoudre le problème.

Il y a deux façons de lancer, dans un environnement, l'exécution de commandes de base pour un autre environnement. La première façon, vue au paragraphe précédent, consiste à "réveiller" l'environnement dormant qui dans la structure est directement au dessus de l'environnement. Ceci peut se faire en spécifiant en zone nom de la commande, le nom de l'environnement dormant. Par exemple, pour exécuter des commandes de l'environnement CP à partir de l'environnement CMS, on procèdera de l'une des deux façons suivantes :

1ère façon :

CP MSG OPERATEUR DEROULEUR LIBRE ?

2ième façon :

CP

MSG OPERATEUR DEROULEUR LIBRE ?

BEGIN

Dans la première forme, l'utilisateur revient dans l'environnement CMS dès que la commande a été exécutée. Par contre, dans la seconde forme, l'environnement CP reste "réveillé" jusqu'à ce que la commande BEGIN soit exécutée. On utilisera donc de préférence cette seconde forme lorsque

l'environnement CP devra être réveillé plusieurs fois de suite. Remarquons cependant que cette forme ne pourra pas être utilisée à l'intérieur d'une macro-commande. En effet, l'exécution de la commande CP (dans la seconde forme) est équivalente à donner le contrôle à l'interpréteur de commandes de cet environnement. Et, comme nous l'avons déjà mentionné auparavant, celui-ci n'a pas la notion de fichiers utilisateur. Donc les commandes qu'il traite ne peuvent provenir que de la console de l'utilisateur et non d'une macro-commande (définie par un fichier); et ce, jusqu'à la rencontre de la commande BEGIN qui fera retourner dans l'environnement CMS. Cet inconvénient est évité avec la première forme puisque le va-et-vient entre les deux environnements se fait pour chaque commande.

La seconde façon d'établir des communications entre deux environnements est l'utilisation d'une pile associée à la console de l'utilisateur. Un élément de cette pile est défini comme étant une commande, c'est à dire une ligne logique identique à celle que l'utilisateur lance à partir de sa console. Une ligne de cette pile est lue dès que l'interpréteur de l'environnement actif demande une lecture de type console. C'est à dire, en fait, qu'il y a lecture à la console seulement lorsque la pile est vide. En conséquence, la façon de communiquer entre deux environnements est de mettre des commandes dans cette pile, puis, dans un second temps, de "réveiller" l'environnement

dormant à qui l'on destine ces commandes. Cette façon sera très utilisée par le macro-processeur. Supposons par exemple, qu'à partir de l'environnement CMS, on utilise le mécanisme d'extension pour générer des commandes pour l'environnement CP. En reprenant la macro-commande déjà utilisée au paragraphe 1.3, avec cette possibilité nous aurons une nouvelle macro-commande qui sera :

```
Nom de macro        IMPRIME
Corps de macro      &STACK CP
                    &STACK SPOOL PRINTER NOHOLD
                    &STACK CHANGE PRINTER DIST &1
                    &STACK CLOSE PRINTER
                    &STACK BEGIN
```

La fonction réalisée par cette macro-commande est exactement la même que celle de la macro-commande décrite au paragraphe 1.3, mais la manière de procéder est différente. L'interprétation de cette macro-commande se traduit par le fait qu'il y a cinq commandes de rangées dans la pile associée à la console de l'utilisateur, par l'intermédiaire de la requête &STACK du langage de contrôle (sa description est donnée au chapitre 2). A la fin de l'interprétation de cette macro-commande, nous sommes toujours sous le contrôle de l'interpréteur du langage de CMS. Celui-ci cherche alors à lire une nouvelle commande au terminal. La pile n'étant pas vide, il prendra ces lignes dans celle-ci. La première

commande trouvée CP, est un "réveil" de l'environnement CP, alors que la dernière (BEGIN) rend cet environnement à nouveau dormant. La différence entre les deux exemples est qu'au paragraphe 1.3 l'interpréteur du langage de commande de CMS reprend le contrôle entre l'exécution de chaque commande; alors qu'ici, il ne le reprend qu'à la fin de l'interprétation de la macro-commande.

Au sujet de l'implémentation de ce mécanisme, nous devons faire une remarque. L'utilisation d'une seule pile, tel que cela est fait actuellement, présente un inconvénient. Nous pouvons en effet trouver dans cette pile des éléments non homogènes, c'est à dire appartenant au langage de commande de plusieurs environnements. Lorsque cette pile sera lue à partir d'un environnement, il pourra alors y avoir incompatibilité, les éléments lus pouvant ne pas appartenir au langage de commande de l'environnement actif. Il serait préférable d'utiliser non plus une seule pile, mais plusieurs piles chacune étant créée à chaque fois qu'un environnement non-actif est rendu actif par exécution d'une commande de transfert. La création et la destruction de ces piles pourraient être intégrées également dans un mécanisme de pile afin de respecter la hiérarchie d'appel des divers environnements. Au sommet de cette pile nous trouverions celle qui correspond à l'environnement actif, et au dessous les piles correspondantes aux environnements dormants dans l'ordre suivant lequel ils ont été appelés.

Une pile associée à la console ne pourrait alors contenir que des éléments homogènes.

Nous pouvons remarquer que ce mécanisme est particulièrement intéressant dans le cas où l'on veut initialiser le contexte d'un environnement. Il suffit en effet avant d'exécuter une commande de transfert, de mettre dans la pile les commandes qui initialiseront l'environnement, puisqu'elles seront lues juste après l'exécution de la commande de transfert.

2 CONTROLE DE L'EXECUTION DES COMMANDES

La manière dont un utilisateur se sert d'un système conversationnel, peut être décomposée en trois temps :

- premièrement, l'utilisateur donne un ordre au système. Il utilise pour cela l'interface de communication entre lui et le système qu'est le langage de commande.
- deuxièmement, le système exécute cet ordre conformément à la description qui en est faite dans le manuel utilisateur.
- enfin, l'utilisateur fait une analyse du résultat pour entreprendre une nouvelle action, ce qui le ramène au premièrement.

Cette façon de procéder pourrait être comparée à celle qu'adopterait une personne qui ferait exécuter ses programmes, écrits en langage machine, pas à pas en se servant des fonctions câblées du panneau ordinateur. Là encore nous aurions trois phases :

- demande d'exécution d'une commande.
- exécution par le hardware de la commande.
- visualisation du mot d'état programme pour connaitre la façon dont s'est déroulée l'exécution.

Nul ne saurait nier l'intérêt des langages de

programmation qui ont permis une utilisation plus souple et plus efficace du langage d'assemblage.

Or, bien que l'interface de communication entre l'utilisateur et le système soit un langage, ce n'est pas à proprement parlé un langage de programmation. Nous n'y trouvons pas en effet les notions classiques que l'on rencontre habituellement, à savoir :

- des variables.
- l'évaluation d'expressions.
- l'exécution conditionnelle d'instructions.

D'où la nécessité d'introduire un nouvel élément afin de pouvoir fournir à l'utilisateur le moyen de construire des programmes de commandes. Outre l'avantage que cela apporte quant à la souplesse d'utilisation du langage de commande, cela permet d'étendre le langage de base; un "programme" de commandes pouvant être considéré comme une nouvelle commande.

Pour résoudre ce problème, nous pouvions envisager deux solutions (réf. HANS) :

- soit utiliser un langage de programmation complet.
- soit utiliser un mécanisme de substitution de chaînes de caractères pour l'écriture de macro-commandes.

Le premier critère dans la détermination de notre choix a porté sur la syntaxe. La syntaxe du langage de commande

est élémentaire. Il nous semblait souhaitable de conserver une certaine unité syntaxique entre le langage de contrôle et le langage de commande afin de libérer l'utilisateur de contraintes inutiles.

D'autre part le mécanisme de substitution associé à une manipulation aisée des chaînes de caractères nous a semblé très attrayant dans un contexte interprétatif qui était le nôtre. C'est donc selon cette seconde approche que nous avons envisagé d'aborder le problème en nous inspirant de travaux de Franck BELVIN et Joel WINNET (réf. IBM CMS) sous CMS.

2.1 CONCEPT DE MACRO

Nous pouvons dire qu'au concept de macro est associée la notion d'extension. D'ailleurs P.J. BROWN définit un macro-processeur comme "un système de programmation conçu pour que l'utilisateur puisse ajouter des facilités de sa propre conception à un système existant" (réf. BROWN). Et une définition de macro permet d'associer à un nom une chaîne de caractères (réf. VIDART 73). Cette chaîne de caractères constitue le corps de la macro. La notion d'extension liée au concept de macro provient de ce qu'à une occurence du nom d'une macro on substitue à ce nom la chaîne

de caractères qui lui est associée.

2.1.1 MACRO-ASSEMBLEURS

Les premiers systèmes d'extension qui ont été proposés, sont les macro-assembleurs. Un langage d'assemblage étendu est constitué du langage de base et d'un ensemble de macros dont l'expansion est une suite d'instructions du langage de base. Le nom de la macro est une instruction du langage étendu. La chaîne associée au nom de la macro (corps de la macro) est constituée d'instructions du langage de base et de nom de macros préalablement définies (réf. ASSABGUI). Dans ces conditions la différence entre le langage de base et le langage étendu est purement syntaxique.

Les facilités offertes par les macro-assembleurs peuvent apparaître, du point de vue utilisateur, analogues à celles de sous-programmes (réf. VIDART 74). La différence fondamentale réside dans ce que certains appellent le "binding time", qui est le moment où un lien est créé entre deux éléments. Ainsi un sous-programme est évalué lors de l'exécution du programme tandis que l'évaluation d'un appel de macro est faite dans une phase préliminaire à cette exécution.

2.1.2 MACRO-PROCESSEUR

Une approche analogue a pu être envisagée pour les langages de haut niveau. C'est McILROY qui, le premier, publia en 1960 un papier dans lequel il rassemblait pour la première fois les idées sur les macro-processeurs, et où il mentionnait que leur usage ne devait pas se confiner aux langages d'assemblage mais pouvait aussi bien être étendu à d'autres langages. Sur la base de ce travail de nombreuses propositions ont été faites par la suite. Nous ne les citerons pas ici et renvoyons les lecteurs intéressés aux bibliographies de BROWN (réf. BROWN) et J. VIDART (réf. VIDART 74). Retenons de ces travaux un aspect particulier développé par LEAVENWORTH qui consiste à appliquer le concept de macro à des extensions syntaxiques pour des langages de haut niveau (réf. LEAVENWORTH). Le but poursuivi est de permettre au programmeur de se définir un langage étendu syntaxiquement à partir du langage de base existant. L'originalité de la méthode réside dans le fait qu'en proposant un système de macros syntaxiques pour des compilateurs dirigés par la syntaxe, le macro-processeur peut être inclus dans le compilateur.

Si ces propositions étaient en grande partie théoriques, notons qu'une implémentation a été réalisée par J. VIDART qui permet de vérifier les idées avancées (réf. VIDART 74).

2.1.3 MACRO-INTERPRETEUR

La différence fondamentale entre les propositions existantes et notre travail provient essentiellement du contexte d'utilisation. Comme nous l'avons vu, dans le cas des macro-assembleurs, ceux-ci se comportaient comme des pré-processeurs pour un assembleur. Et l'évaluation d'un appel de macro était dans une phase préliminaire à l'assemblage. Dans notre cas, le contexte dans lequel nous nous sommes placés est un contexte interprétatif. On ne peut donc pas dissocier l'évaluation d'une macro de l'exécution des commandes de bases. Une dialectique s'établit donc entre l'interpréteur du langage de base et le macro-interpréteur en ce sens que le processus d'évaluation d'une ligne de la macro peut être modifié par les résultats de l'exécution de la commande de base précédente.

D'autre part, nous ne pouvons pas dire non plus que nous sommes en présence d'un mécanisme analogue aux sous-programmes. En effet, dans un sous-programme, tout le texte est connu du processeur, en l'occurence un compilateur. Dans notre cas nous trouvons des mécanismes de substitution de chaîne de caractères et des instructions de contrôle, inconnus de l'interpréteur du langage de base, qui sont typiquement des éléments de type macro.

2.2 CONSIDERATIONS SYNTAXIQUES

Il y a principalement trois points sur lesquels des considérations d'ordre syntaxique peuvent être faite :

- la forme syntaxique de l'appel d'une macro.
- la syntaxe des paramètres formels à l'intérieur de la macro.
- la forme syntaxique du langage de commande à l'intérieur de la macro.

L'intérêt du mécanisme de macro pour un langage de commande est, entre autre, de permettre à chaque utilisateur de personnaliser le langage de commandes. Nous pouvons penser que celui-ci fera des appels excessivement fréquents à ce mécanisme (réf. LEROUDIER). Les considérations d'ordre syntaxique sont importantes en ce sens que les contraintes d'utilisation doivent être minimales. Nous aborderons cette question pour chacun des trois points ci-dessus.

2.2.1 SYNTAXE DE L'APPEL

Ainsi que nous l'avons déjà décrit en 1.1, le langage de commande pour le système est constitué d'un ensemble de commandes dont la syntaxe de chacune est la suivante :

nom-de-commande liste de paramètres

C'est à dire que chaque commande est constituée d'un ensemble de mots, le premier étant le nom de la commande et les autres une liste de paramètres.

Le but que nous poursuivons étant d'étendre le langage de commande par un macro-mécanisme, il était souhaitable de conserver pour l'appel d'une macro la même syntaxe que pour une commande. Nous avons donc gardé la même forme que dans le processeur EXEC de CMS (réf. IBM EXEC) c'est à dire :

EXEC nom-de-macro liste de paramètres

Une différence existe en ce qui concerne la liste de paramètres d'une macro-commande et d'une commande de base. Pour les commandes de base, les paramètres sont des mots séparés par des blancs. Dans une macro-commande, la différence provient de ce que nous avons introduit la notion de chaîne de caractères. Le caractère blanc constitue donc un séparateur entre les arguments de la macro-commande que s'il n'appartient pas à un élément de type chaîne. Exemple

IMPRIME LETTRE 'Mr. A DUPOND'

est une macro-commande de nom IMPRIME qui a deux arguments.

2.2.2 SYNTAXE DES PARAMETRES FORMELS

Les paramètre formels apparaissent dans le corps d'une macro-commande pour indiquer l'endroit où leur seront substitués les arguments. La syntaxe utilisée pour repérer ces paramètres est ce que nous appellerons une désignation par numéro.

Avec cette méthode, un paramètre formel est représenté par un nombre précédé d'un caractère spécial &. Le nombre spécifié représente la position de l'argument dans la ligne d'appel qui sera associé au paramètre formel. Ainsi en reprenant l'exemple de la macro-commande IMPRIME ci-dessus, au paramètre formel &2 on associera l'argument Mr. A DUPOND.

2.2.3 SYNTAXE DU LANGAGE DE CONTROLE

Ainsi que nous l'avons déjà mentionné, ce point particulier nous a paru important du fait de l'utilisation fréquente de ce mécanisme que nous étions en mesure d'attendre.

Le premier point est que dans une macro-commande on rencontre deux types de langage. Il y a d'une part le langage de commande; d'autre part il y a le langage de contrôle. La syntaxe du premier est fixée par le manuel de l'utilisateur. C'est à celle du second que nous avions à nous intéresser. En choisissant pour le langage de contrôle

un langage de programmation complet, nous étions conduits à introduire une seconde syntaxe. Ceci pouvait également amener des structures ennuyeuses pour l'utilisateur comme par exemple la déclaration de variables. De plus la syntaxe du langage de commande pouvait se trouver modifiée par rapport à celle qui est en usage lorsque les commandes sont exécutées à partir de la console.

Illustrons ces idées par un exemple. A la figure 4, nous avons écrit deux macro-commandes, strictement équivalentes sur le plan fonctionnel. En 4-a, le langage de contrôle qui est utilisé est PL/I, alors qu'en 4-b c'est celui du processeur EXEC de CMS (réf. IBM EXEC). La comparaison de ces deux exemples nous conduit à faire plusieurs constations. La première c'est que la version EXEC est beaucoup plus courte. Deuxièmement l'homogénéité entre le langage de commande et le langage de contrôle est plus grande, donc l'utilisation est plus simple. Enfin les commandes du langage de base ont la même syntaxe qu'à la console. Le choix d'un tel langage comme langage de contrôle est donc manifestement plus attrayant.

Un autre élément important dans un langage spécialisé comme le langage EXEC est ce que nous appellerons la "syntaxe dynamique". Considérons par exemple la séquence d'instructions suivantes :

```
LISTSORT: PROC ( PARM1, PARM2, CODE );
DECLARE (PARM1, PARM2) CHAR(*) VARYING, CODE BINARY FIXED;
DECLARE ESCAPE ENTRY EXTERNAL (CHAR(*) VARYING,
                        BINARY FIXED);
DECLARE RETCODE BINARY FIXED;
     CALL ESCAPE ( 'ASSEMBLE '//PARM1//' (NOTERM'//PARM2,
                                   RETCODE);
     IF RETCODE ≠ 0 THEN DO;
          CODE = 1;
          RETURN;
     END;
     CALL ESCAPE ( 'PRINT '//PARM1, RETCODE);
     CODE = 0;
END;
```

4-a Version PL/I

```
ASSEMBLE &1 (NOTERM &2
&IF &RETCODE NE 0 &EXIT 1
PRINT &1
```

4-b Version EXEC

Figure 4 Deux types de Macro-Commande

(1) &Y=A

(2) &X&Y=3

(3) &Y=B

(4) &X&Y=TOTO

A la ligne (2), la valeur 3 sera affectée à la variable de nom &XA; tandis qu'à la ligne (4), c'est la variable de nom &XB qui prendra la valeur TOTO. Cette particularité, qui apporte une grande souplesse d'utilisation au langage, est dûe au mécanisme de substitution qui consiste à remplacer dans un mot le nom d'une variable par sa valeur; mécanisme qui n'existe pas dans les langages de haut niveau.

En conclusion, nous pouvons dire que ces quelques considérations syntaxiques sont toutes à l'avantage d'un langage spécialisé comme celui d'EXEC.

2.3 FACILITES DU MACRO-LANGAGE

D'une façon générale, les systèmes de macro ont un langage de programmation interne qui d'une part peut donner des ordres au macro-processeur, et d'autre part permet de définir et de manipuler certaines entités. Nous envisageons dans cette partie les facilités offertes par le prototype que nous avons réalisé.

2.3.1 CHAINES DE CARACTERES

Ainsi que nous l'avons déjà mentionné, le langage de commande est un langage à mots. C'est à dire que la structure de base du langage est le mot. Il était nécessaire d'avoir les moyens, pour l'utilisateur, de les manipuler de façon efficace. Le premier point important dans ce domaine concerne leur rigidité. Et, il nous a semblé qu'imposer une longueur fixe aux chaînes de caractères représentait une contrainte importante pour avoir une manipulation souple et efficace. D'autre part, nous pourrions envisager par exemple une description de fichier faisant intervenir une chaîne de caractères pour les désigner (description arborescente) et un mécanisme de code de retour qui soit généralisé (chaîne assimilable par le macro-mécanisme). C'est pourquoi, dans le prototype que nous avons réalisé, nous avons introduit les chaînes de caractères de longueur variable.

Pour cela, nous avons tout d'abord défini la notion de chaîne de caractères. Par définition, nous dirons qu'une chaîne de caractères est un ensemble de caractères compris entre deux apostrophes. Une chaîne de caractères peut constituer un mot à elle seule ou faire partie d'un mot. Par exemple

'Mr. DUPONT' et 'DMS'.&A

sont deux mots. Le premier est seulement constitué d'une chaîne de caractères, tandis que le second est composé d'une

chaîne concaténée à un nom de variable.

Cette notion de chaîne de caractères de longueur variable acquiert toute son importance dans la mesure où l'idée directrice est d'avoir comme interface de communication entre le système et l'utilisateur un langage de base réduit à quelques primitives et un macro-mécanisme d'extension pour compléter ce langage. Nous devons rechercher pour le langage de base une syntaxe simple. Et souvent pour de tels langages, elle se résume à une juxtaposition de mots. Nous pouvons par exemple envisager une méthode d'accès aux fichiers utilisateur qui soit arborescente. C'est en particulier ce qui a été envisagé pour la gestion de fichiers de CMS+ (réf. LE HEIGET), dans lequel EXEC+ devrait être intégré. Une désignation d'un fichier sera alors :

A1.A2.A3.A4.A5

où les Ai désignent le sélecteur de niveau i dans l'arborescence. D'où pour l'utilisateur une manipulation fréquente des chaînes de caractères.

Nous voyons donc la nécessité d'introduire dans le macro-mécanisme les outils nécessaires pour une utilisation souple et efficace des chaînes. C'est pourquoi, en plus de la notion de chaîne de longueur variable, nous avons introduit une fonction de concaténation qui soit généralisée et qui fonctionne parallèlement au mécanisme de

substitution. L'opérateur de concaténation est le point (.) et il peut apparaitre à l'intérieur de tout mot. La description complète de ces mécanismes est faite au chapitre II paragraphes 3.3 et 4.3. Ainsi les facilités offertes dans notre prototype sont comparables, en ce qui concerne la manipulation des chaînes de caractères, à celles du langage SNOBOL (réf. SNOBOL) et rendent le langage de contrôle souple et efficace.

2.3.2 VARIABLES INTERNES

Le langage de contrôle du prototype réalisé autorise la définition et la manipulation de variables. L'utilisateur peut définir deux sortes de variables :

- les variables locales.
- les variables globales.

Une variable est dite locale lorsque sa portée d'utilisation correspond au corps de la macro-commande dans laquelle elle est définie. Par contre les variables globales sont accessibles dans toute macro-commande de niveau de récursion supérieur ou égal à 1. La forme externe d'une variable est &XYYY... où X est un caractère alphabétique. Pour les variables globales X doit être le caractère G. La chaîne de caractères représentant le nom d'une variable peut être de longueur quelconque.

Les variables définies par l'utilisateur peuvent être de type caractère ou de type numérique. La conversion de type est automatique et dépend de la sémantique. Les seules valeurs que peut prendre une variable de type numérique appartiennent à l'ensemble des entiers relatifs. Quant aux opérations sur ces variables elles sont limitées à l'addition et à la soustraction. En ce qui concerne les variables de type caractère elles peuvent prendre elles aussi des valeurs de longueur quelconque.

2.3.3 INSTRUCTIONS DE CONTROLE

L'expérience ayant montré le bien-fondé du choix qui a été fait pour les instructions de contrôle du langage EXEC, nous les avons reprises en majorité. D'autant plus que nous souhaitions que notre prototype reste compatible avec les anciennes versions afin d'éviter que les utilisateurs n'aient à réécrire leurs macro-commándes.

Comme instructions de contrôle, nous trouvons tout d'abord les instructions "conditionnelles". Ce sont les requêtes &IF et &LOOP qui permettent d'exécuter une ou plusieurs commandes selon qu'une condition est satisfaite ou non. Dans cette catégorie nous classons également la requête &ERROR qui est l'équivalent du "ON ERROR" de PL/I. C'est à dire qu'on exécutera la commande spécifiée en zone paramètre

de la requête, chaque fois qu'il y aura eu une erreur à l'exécution d'une commande de base.

Parmi les instructions de contrôle nous trouvons une requête &GOTO qui autorise le transfert de contrôle. Dans ce cas, il y a branchement à la commande dont le premier mot est l'étiquette spécifiée. Une étiquette est une chaîne de caractères commençant par le symbole - et ayant au plus 10 caractères. Notons que toutes les lignes d'une macro-commande peuvent commencer par une étiquette; c'est à dire aussi bien une commande du langage de base qu'une instruction d'affectation ou une requête de contrôle.

Parmi l'ensemble des requêtes du langage de contrôle une place particulière est occupée par la requête &READ. La particularité de cette requête provient de ce qu'on a un macro-mécanisme pour un système conversationnel. Il était important de conserver l'aspect interactif pour les macro-commandes. Ainsi, les lignes lues à la suite de l'interprétation de cette requête sont considérées par le macro-mécanisme comme si elles appartenaient à la macro-commande. Nous pouvons donc exécuter de façon interactive aussi bien des commandes du langage de base que des requêtes de contrôle.

C H A P I T R E I I

D E S C R I P T I O N E X T E R N E
D E E X E C +

1 CREATION D'UNE MACRO-COMMANDE EXEC

Comme pour tout fichier que l'on veut créer dans le système CMS, l'utilisateur dispose de trois moyens pour créer une macro-commande :

- soit il perfore des cartes qui seront lues sur le lecteur réel par le système CP et placées ensuite dans le lecteur virtuel de la machine utilisateur.
- soit il utilise l'éditeur de CMS pour créer un fichier de type EXEC. Pour cela il se sert de la commande EDIT dont le format est

  EDIT nom-de-fichier EXEC

  Cette commande ayant été lancée, il rentrera dans le mode INPUT. Les lignes rentrées à la console constitueront alors le corps de la macro-commande.
- soit l'utilisateur crée un fichier de type EXEC par programme.

De ces trois moyens, c'est le second qui est utilisé le plus souvent du fait de sa souplesse et de sa commodité d'emploi.

Remarquons que l'option EXEC de la commande CMS LISTFILE, permet de créer automatiquement une macro-commande. Celle-ci a pour nom CMS et une ligne a le format suivant :

&1 &2 ligneN

où ligneN a la même forme que celle de la ligne qui aurait été imprimée au terminal à l'exécution de la commande LISTFILE sans l'option EXEC. Ainsi par exemple la commande

LISTFILE * ASSEMBLE (EXEC

se traduira par la création d'un fichier CMS EXEC A1 dont les lignes seront de la forme &1 &2 filename ASSEMBLE A1, filename étant le nom d'un des fichiers de type ASSEMBLE appartenant à l'utilisateur.

Le nom de la macro-commande créée est le même que celui du fichier.

2 APPEL D'UNE MACRO-COMMANDE

Pour demander l'exécution d'une macro-commande, l'utilisateur rentre la commande EXEC suivie du nom de la macro-commande et éventuellement d'une liste de paramètres. Cependant, lorsque l'appel se fait à partir de l'environnement CMS le mot de commande EXEC peut être omis. En effet dans ce cas, le module qui lit la commande au terminal regarde tout d'abord s'il n'y a pas de fichier de type EXEC dont le nom soit identique au premier mot de la commande. Si tel est le cas, la commande est passée à l'interpréteur EXEC. Ceci implique que si une commande CMS a le même nom qu'une macro-commande il peut y avoir conflit d'appel. Pour éviter cela on définit un ordre de recherche; par défaut c'est la macro-commande qui est prioritaire et

donc qui sera lancée. Il est néanmoins possible de modifier cette priorité par une commande.

Par contre, dans la cas où une macro-commande EXEC est lancée à partir d'un programme, le mot EXEC ne doit pas être omis.

3 CARACTERISTIQUES GENERALES DES MACRO-COMMANDES

3.1 LIGNES D'UNE MACRO-COMMANDE

Une macro-commande est un fichier CMS de type EXEC qui peut contenir deux sortes d'éléments. On rencontre d'une part les lignes appartenant au langage de commande CMS. D'autre part on trouve les lignes appartenant au langage de contrôle EXEC.

3.1.1 COMMANDES CMS

Une ligne d'une macro-commande est une commande CMS si le premier mot de cette ligne ne commence pas par le caractère &. Si la ligne commence par une étiquette, on ne tient pas compte de cette étiquette.

Notons que les commandes propres au langage de commande de CP appartiennent à cette classe. Elles s'en différencient

par le fait que la commande est préfixée par le mot CP.

Exemple :

CP MSG OPERATOR ATTACHER LA BANDE 235

3.1.2 LANGAGE DE CONTROLE EXEC

Les lignes d'une macro-commande qui appartiennent à cette catégorie peuvent être classées en quatre groupes :

a) les lignes nulles.

Ces lignes ne contiennent aucun mot.

b) les lignes commentaires.

Le premier caractère de ces lignes doit être un astérisque. Elles servent de commentaire à l'intérieur de la macro-commande et sont ignorées au moment de l'interprétation.

c) les instructions de contrôle.

Une ligne d'une macro-commande est une instruction de contrôle si le premier mot (non compris l'étiquette) est un mot réservé du langage de contrôle EXEC.

Remarquons que dans certains cas une ligne peut être une donnée pour une instruction de contrôle. C'est par exemple le cas des lignes comprises entre les commandes &BEGTYPE et &END.

d) les instructions d'affectation.

Une ligne d'une macro-commande est une instruction

d'affectation si le premier mot est le nom d'une variable et le deuxième élément de la ligne est le signe =.

Notons que toute ligne d'une macro-commande excepté les ligne commentaires peuvent commencer par une étiquette. Une étiquette est un mot dont le premier caractère est un tiret (-, suivi d'au plus dix caractères.

3.2 VARIABLES

Une propriété du langage de contrôle EXEC est d'être également un langage de manipulation de variables. A l'intérieur d'une macro-commande on rencontre deux sortes de variables :

- les variables définies par l'utilisateur.
- les variables spéciales, propres à l'interpréteur EXEC que l'utilisateur ne peut accéder qu'en lecture.

La forme externe d'une variable utilisateur est une chaîne de caractères de longueur variable dont le premier caractère est & et le second, un caractère alphabétique. Ces variables peuvent être de type alphabétique ou numérique. Ce n'est qu'au moment de l'interprétation qu'il y aura conversion éventuelle d'un type dans l'autre. Cette

conversion de type est automatique et elle est déterminée par la sémantique de la commande dans laquelle apparait la variable. Pour les variables arithmétiques, seules sont autorisées les opérations arithmétiques simples (addition et soustraction) et la valeur prise par une variable ne peut être qu'un entier relatif.

L'utilisateur a la possibilité de se définir deux sortes de variables :

- les variables locales.
- les variables globales.

Une variable est une variable locale lorsque sa portée d'utilisation est locale à la macro-commande dans laquelle elle est définie.

Une variable est une variable globale lorsque son domaine d'utilisation est l'ensemble des macro-commandes de niveau de récursion supérieur ou égal à un.

De façon externe ces deux types de variables se distinguent par le fait que les variables globales ont un nom qui doit commencer par les caractères &G.

Les variables spéciales, décrites ultérieurement, ont une syntaxe et un sens pré-établis. Seul l'interpréteur peut les modifier. La conversion automatique de type s'applique également pour ces variables selon leur contexte d'utilisation.

3.3 CONCATENATION

La concaténation est une fonction interne dont l'utilisation a été généralisée. L'opérateur de concaténation est le point. Il peut être utilisé à l'intérieur de tout mot. Cependant, il perd sa valeur d'opérateur s'il appartient à une chaîne de caractères; c'est à dire s'il se trouve entre deux apostrophes.

L'opération de concaténation a lieu en parallèle avec l'opération de substitution. Ceci donne au langage une grande souplesse d'utilisation. Il est ainsi possible de manipuler des éléments de tableau. Un élément d'un tableau à deux dimensions sera désigné par

&A.&I.&J

dans lequel les variables &I et &J seront les indices

Remarquons que l'opérateur de concaténation n'est pas nécessaire si le transfert de chaîne est fait par le mécanisme de substitution. Ainsi par exemple l'élément DMS.&I est strictement équivalent à l'élément DMS&I. Par contre cet opérateur est nécessaire pour les concaténations à droite comme par exemple dans l'élément &A.TOTO.

4 MANUEL DE L'UTILISATEUR DE EXEC+ SOUS CMS

4.1 MOT - DELIMITEUR

Dans une macro-commande l'élément de base qui est manipulé est le mot. Par convention, un mot est une suite de caractères, de longueur variable, comprise entre deux délimiteurs. Un délimiteur est le caractère blanc ou l'un des caractères suivants :

+ - = ( ) :

La notion de mot est donc la même que pour les langages naturels ou pour les langages de formulation mathématique où elle est équivalente à la notion d'élément. Ainsi la ligne

```
&A&I=&SUBSTR &VAR (&DEBUT : &LONG)
```

est constituée de cinq mots.

4.2 CHAINES DE CARACTERES

Par convention une chaîne de caractères est une suite de caractères comprise entre deux apostrophes. Remarquons que le caractère apostrophe appartient lui-même a la chaîne de caractères s'il est doublé. Cette notion de chaîne de caractères permet d'affiner celle de mot qui acquiert toute son importance dans le mécanisme de substitution. Ceci provient du fait que tout délimiteur perd son caractère

particulier s'il fait partie d'une chaîne de caractères. On peut d'autre part trouver dans un mot une chaîne de caractères. Soit par exemple la ligne :

&TYPE 'LES ELEMENTS IMPRIMES SONT' 'DM'.&FIC

Cette ligne est composée de trois mots dont le second est une chaîne de caractères et le troisième composé d'une partie chaîne et d'une partie nom de variable.

4.3 SUBSTITUTION

Le mécanisme de substitution constitue l'une des caractéristiques essentielles du macro-processeur EXEC+. C'est en effet ce mécanisme qui est à la base du processus de manipulation des mots. Contrairement à ce qu'offre la plupart des langages de programmation, ce mécanisme donne le moyen d'avoir une syntaxe dynamique. La valeur syntaxique d'un mot n'est en effet définie qu'après qu'il soit passé par ce mécanisme qui est utilisé avant chaque interprétation. Le principe de fonctionnement de ce mécanisme est le suivant :

Tout mot est analysé de la droite vers la gauche. Si l'on trouve une apostrophe indiquant que l'on a une chaîne de caractères, les caractères faisant partie de cette chaîne ne sont pas analysés. Une substitution a lieu lorsque l'on détecte un caractère &. A ce moment, lui, ainsi que tous les

caractères à sa droite jusqu'à l'opérateur de concaténation (un point) s'il apparait dans le mot, ou jusqu'au dernier caractère, sont considérés comme étant le nom d'une variable. On remplace alors dans le mot les caractères constituants le nom de la variable par ceux représentant la valeur de celle-ci. Nous obtenons ainsi un nouveau mot. L'analyse de ce mot reprend alors à partir du caractère immédiatement à gauche du caractère & sur lequel elle s'était arrêtée. Le fait de ne pas rechercher un caractère & dans la partie substituée évite au mécanisme de boucler indéfiniment. Cette procédure d'analyse est répétée jusqu'au dernier caractère du mot le plus à gauche. Néanmoins lorsque le mot est le nom d'une variable se trouvant à gauche du signe égal dans une instruction d'affectation ce mécanisme de substitution s'arrête à l'avant dernier caractère. Il convient en effet dans ce cas de conserver le nom d'une variable et non sa valeur.

Illustrons sur un exemple le fonctionnement de ce mécanisme :

AVANT SUBSTITUTION	APRES SUBSTITUTION
&A=1	&A=1
&B=2	&B=2
&X&B='MONFICHIER'.&A	&X2=MONFICHIER1
&A&B=&A+&B	&A2=1+2
&Y=&A&B+&A.&B	&Y=&A2+1.2 soit
	&Y=3+12

4.4 DESCRIPTION DE LA COMMANDE EXEC

Objet : la commande EXEC a pour but de lancer l'exécution d'une macro-commande. Cette macro-commande contient soit des commandes CMS, soit des requêtes de contrôle EXEC. Celles-ci sont contenues dans un fichier de type EXEC.

Format

```
-----------------------------------------------
I                                             I
I        EXEC  filename  arg1 arg2.....       I
I                                             I
-----------------------------------------------
```

Utilisation :

filename indique le nom du fichier EXEC dont on désire que les lignes soient interprétées.

arg1 arg2 .... constituent la liste de paramètres qui sera passée à la macro-commande. Une macro-commande peut avoir jusqu'à trente paramètres. On les désigne sous le nom d'argument. Chaque argument est représenté par un caractère & suivi d'un entier compris entre 1 et 30.

Le passage des paramètres se fait par position. c'est à dire que, au moment de l'appel, &1 prend la valeur arg1, &2 prend la valeur arg2 etc...

Si le signe % est utilisé à la place de argN, l'argument correspondant &N ne sera pas initialisé.

Si le fichier EXEC contient plus d'arguments qu'il n'y en a de fourni au moment de l'appel de la macro-commande, ceux qui ont un numéro d'ordre supérieur à celui du dernier de la ligne d'appel ne seront pas initialisés.

Un argument de la ligne d'appel est un mot compris entre deux blancs. Le caractère blanc peut lui-même être banalisé; il suffit pour cela de l'encadrer d'apostrophes.

Réponse : -s'il n'existe pas de fichier EXEC ayant le même nom que celui indiqué au moment de l'appel, on revient dans l'environnement d'appel avec un code d'erreur 28.

TOO MANY ARGS

Le nombre d'arguments passés dans la ligne d'appel est supérieur à 30.

Exemple : EXEC PRINT COURRIER 'MR. DUPONT'

Le nom de la macro-commande dont on demande l'exécution est PRINT. Les arguments passés à cette macro-commande sont COURRIER et MR. DUPONT.

LISTE DES REQUETES DE CONTROLE EXEC

&ARGS	redéfinit dynamiquement les arguments.
&BEGPUNCH	sert à perforer des lignes.
&BEGSTACK	met des lignes dans la pile de la console.
&BEGTYPE	imprime des lignes au terminal.
&CONTINUE	aucune action.
&CONTROL	permet une trace console de l'interprétation.
&END	ferme les blocs &BEG...
&ERROR	exécution suite à une erreur CMS.
&EXIT	fait sortir de la macro-commande.
&GOTO	autorise les ruptures de séquence.
&IF	exécution conditionnelle d'une instruction.
&LOOP	permet d'exécuter plusieurs fois un même nombre de lignes.
&PUNCH	perforation d'une ligne.
&READ	lance des lectures console.
&SKIP	saut de lignes dans la macro-commande.
&SPACE	saut de lignes au terminal.
&STACK	range une ligne dans la pile de la console.
&TIME	permet de connaitre le temps virtuel et total consommés.
&TYPE	impression d'une ligne au terminal.

4.5 DESCRIPTION DES REQUETES DU LANGAGE DE CONTROLE EXEC

&ARGS

Objet : la requête de contrôle &ARGS permet à l'utilisateur de redéfinir un ou plusieurs arguments de façon dynamique.

Format :

```
-----------------------------------------------
I                                             I
I    &ARGS  arg1 arg2 ..... argN              I
I                                             I
-----------------------------------------------
```

Utilisation :

La commande &ARGS est utilisée pour redéfinir les arguments &1, &2, ..., &N avec les valeurs spécifiées par arg1, arg2, ..., argN. Comme lors de l'appel de la macro-commande, le symbole % permet de ne pas redéfinir un argument; ce qui est équivalent à lui affecter la chaîne nulle. A l'exécution de cette requête, on réinitialise la variable spéciale &INDEX.

Exemple :

```
&ARGS ASSEMBLE TAUX
-ETIC &1 &2 &3
&ARGS PRINT TAUX LISTING
```

&IF &INDEX EQ 2 &GOTO -ETIC

Au premier passage sur la ligne -ETIC, il y aura assemblage du fichier TAUX et au second passage, impression de ce même fichier. Le processus ne boucle pas puisqu'au test final on aura &INDEX qui vaudra 3.

&BEGPUNCH

Objet : la requête &BEGPUNCH permet d'envoyer sur le perforateur de la machine virtuelle une ou plusieurs lignes. Les lignes perforées sont celles qui sont comprises entre les requêtes &BEGPUNCH et &END.

Format :

```
---------------------------------------------
I                                           I
I         &BEGPUNCH     ( ALL )             I
I                                           I
I            ligne1                         I
I            ligne2                         I
I              .                            I
I              .                            I
I            ligneN                         I
I                                           I
I         &END                              I
I                                           I
---------------------------------------------
```

Utilisation :

Les lignes ligne1, ligne2, ..., ligneN sont envoyées dans le perforateur virtuel de la machine de l'utilisateur. La remarque importante pour cette commande est que les lignes à perforer ne passent pas par le mécanisme de substitution. C'est à dire qu'elles sont perforées dans la forme sous laquelle elles apparaissent dans la macro-commande. Les lignes avant d'être perforées sont normalement tronquées à partir de la colonne 72, sauf si l'option ALL est spécifiée. Dans ce cas, elles sont perforées sur 80 colonnes.

La requête &END indique la fin du bloc de lignes.

Réponse : AN &BEG... BLOCK IS NOT CLOSED

Dans ce cas il y a eu une fin de fichier avant de trouver un &END.

&BEGSTACK

Objet : cette requête est la tête d'une liste de lignes destinées à être placées dans la pile associée à la console de l'utilisateur. Les lignes à ranger sont celles qui sont comprises entre la requête &BEGSTACK et la requête &END.

Format :

```
-----------------------------------------------
I                                             I
I     &BEGSTACK  ( LIFO )    ( ALL )          I
I                ( FIFO )                     I
I                                             I
I         ligne1                              I
I           .                                 I
I           .                                 I
I         ligneN                              I
I                                             I
I     &END                                    I
-----------------------------------------------
```

Utilisation :

Par cette requête l'utilisateur a la possibilité de mettre dans la pile associée à sa console un certain nombre de lignes. Celles-ci seront lues dès qu'un processus demandera une lecture de type console. Les lignes ne passent pas par le mécanisme de substitution et elles sont rangées telles qu'elles apparaissent à l'intérieur de la macro-commande.

Il y a deux façons de ranger les lignes dans la pile : suivant le mode FIFO ou le mode LIFO. Dans le premier mode, la ligne rangée la première

sera la première lue. Dans le second, c'est la dernière rangée qui sera la première lue. Par défaut, le mode d'utilisation est FIFO.

Les lignes sont normalement tronquées à partir de la colonne 72 sauf si l'on spécifie l'option ALL. Dans ce cas, elles ont une longueur de 130 caractères. La liste des lignes à ranger dans la pile se termine par la requête &END.

Réponse : AN &BEG... BLOCK IS NOT CLOSED.

Il y a eu dans ce cas une fin de fichier avant de trouver la ligne &END.

Exemple :

```
&BEGSTACK
TOP
LOCATE /BALR 14,15/
CHANGE /BALR 14,15/SVC 202/
FILE
&END
EDIT DMSLINE ASSEMBLE
```

Les quatre requêtes d'édition (TOP, LOCATE, CHANGE et FILE) sont mises dans la pile. On appelle ensuite l'éditeur pour le fichier DMSLINE. Il y aura exécution automatique de ces quatre requêtes d'édition.

&BEGTYPE

Objet : cette requête a pour but d'imprimer des lignes sur la console de l'utilisateur. Les lignes à imprimer sont celles qui sont comprises entre &BEGTYPE et &END.

Format :

```
---------------------------------------------
I                                           I
I    &BEGTYPE    ( ALL )                    I
I                                           I
I        ligne1                             I
I          .                                I
I          .                                I
I        ligneN                             I
I                                           I
I    &END                                   I
I                                           I
---------------------------------------------
```

Utilisation :

Cette requête permet d'imprimer un certain nombre de lignes sur la console de l'utilisateur. Le mécanisme de substitution n'est pas appliqué pour ces lignes. Leur forme à la console est donc la même qu'à l'intérieur de la macro-commande. Ces lignes sont normalement tronquées à partir de la colonne 72. Si l'option ALL est spécifiée, leur longueur est portée à 130 caractères. La liste des lignes doit se terminer par la requête &END.

Réponse : AN &BEG... BLOCK IS NOT CLOSED.

On a rencontré une fin de fichier avant d'avoir

trouvé la requête &END.

Exemple : &BEGTYPE ALL

Les paramètres que vous avez passés dans la ligne d'appel

ne sont pas corrects; veuillez les respécifier.

&END

&CONTINUE

Objet : cette requête ne conduit à aucune action.

Format :

```
-----------------------------------------------
I                                             I
I    &CONTINUE                                I
I                                             I
-----------------------------------------------
```

Utilisation :

On utilise généralement cette requête comme support d'une étiquette pour fournir une adresse de branchement pour les requêtes &IF, &GOTO ou &LOOP.

Exemple : &LOOP -FIN 3

&A=&A+2

-FIN &CONTINUE

&CONTROL

Objet : cette requête renseigne l'interpréteur sur la façon de fournir une trace de l'exécution de la macro-commande, sur la console de l'utilisateur.

Format :

```
-----------------------------------------------
I                                             I
I    &CONTROL  ( OFF )   ( TIME )             I
I              ( ERROR )  ( NOTIME )          I
I              ( CMS )                        I
I              ( ALL )                        I
I                                             I
-----------------------------------------------
```

OFF les commandes CMS et les requêtes de contrôle EXEC ne sont pas imprimées sur la console. Il en est de même pour les codes erreur éventuels.

ERROR seules les commandes CMS dont l'exécution ne s'est pas déroulée correctement sont imprimées ainsi que le code d'erreur s'il est non nul.

CMS toutes les commandes CMS contenues à l'intérieur de la macro-commande sont listées sur le terminal de l'utilisateur. Seuls les codes d'erreur non nuls sont imprimés.

ALL l'utilisateur a une trace complète des instructions exécutées et des codes d'erreur non nuls

TIME en même temps que l'impression d'une commande CMS on a la valeur de l'heure à laquelle cette commande a été lancée. Cette option n'a de sens qu'en conjugaison avec les options ERROR, CMS ou ALL.

NOTIME il n'y a pas impression de l'heure lors de l'exécution des commandes CMS.

Utilisation :

Cette requête permet à l'utilisateur d'avoir une trace appropriée de la façon dont la macro-commande s'est déroulée. Les options par défaut pour cette requête sont CMS et NOTIME. Chaque option reste inchangée jusqu'à une nouvelle spécification.

Exemple : &CONTROL ERROR TIME

Toutes les commandes CMS dont l'exécution a entrainé une erreur seront imprimées à la console avec l'heure et le code d'erreur. On aura par exemple :

09:47:37 MOVEFILE

--* R(-00003) *-*-*

&END

Objet : la requête &END termine une liste de lignes commençant par &BEGPUNCH, &BEGSTACK ou &BEGTYPE.

Format :

```
-----------------------------------------------
I                                             I
I     &END                                    I
I                                             I
-----------------------------------------------
```

Utilisation :

Les lignes comprises entre &BEGPUNCH, &BEGSTACK ou &BEGTYPE et &END sont prises comme un groupe de lignes sur lesquelles il n'y a pas de substitution de faite. Ces lignes sont perforées, empilées ou imprimées dans la même forme que celle dans laquelle elles apparaissent à l'intérieur de la macro-commande. Pour que la requête &END soit prise en compte, il est nécessaire qu'elle commence en colonne 1. Sinon, elle est considérée comme faisant partie du groupe de lignes.

Réponse : AN &BEG... BLOCK IS NOT CLOSED

il y a eu une fin de fichier avant de trouver &END.

&ERROR

Objet : cette requête permet d'avoir l'exécution automatique d'une instruction après chaque commande CMS dont l'exécution a provoqué une erreur.

Format :

```
-----------------------------------------------
I                                             I
I   &ERROR  action                            I
I                                             I
-----------------------------------------------
```

Utilisation :

L'utilisation de cette requête est l'équivalent du 'ON CONDITION' pour certains langages de haut niveau. C'est à dire qu'après l'exécution d'une commande CMS dont le code de retour est non nul, c'est la commande "action" qui devient la prochaine commande à exécuter.

L' "action" spécifiée peut être toute commande exécutable. Si l' "action" est elle-même une commande CMS dont l'exécution se termine par une erreur, on ne tient pas compte de cette condition permanente; le processus d'interprétation de la macro-commande en cours est arrêté.

Après l'exécution de l' "action", l'interprétation reprend avec la ligne qui suit la commande qui a provoqué l'erreur; à moins que l' "action" ne soit une commande de transfert.

La commande "action" est analysée avant chaque interprétation et peut donc prendre des significations différentes.

Par défaut la valeur de l' "action" est la requête &CONTINUE.

Réponse : ERROR IN &ERROR ACTION

L' "action" est une commande CMS dont l'exécution a provoqué une erreur.

Exemple : &ERROR &EXIT &RETCODE

Chaque fois qu'il y aura une erreur dans une commande CMS, on sortira de la macro-commande avec en indication la valeur du code d'erreur.

&ERROR &TYPE 'ERREUR A L''EXECUTION DE LA COMMANDE NO.' &LINENUM

A chaque erreur il y aura impression d'un message sur la console de l'utilisateur indiquant la ligne où s'est produit cette erreur.

&EXIT

Objet : cette requête permet de sortir de la macro-commande en cours d'interprétation, avec la possibilité de spécifier un code numérique.

Format :

```
-----------------------------------------------
I                                             I
I    &EXIT     ( code-de-retour )             I
I              ( 0 )                          I
I                                             I
-----------------------------------------------
```

Utilisation :

On utilise la requête &EXIT pour passer d'un niveau de récursion au niveau inférieur, ou pour retourner dans l'environnement appelant si l'on se trouve au niveau 1.

Si aucun code de retour n'est fourni, la valeur prise par défaut est 0. Pour ce code de retour la convention adoptée par CMS est respectée; c'est à dire que ce code de retour est passé par l'intermédiaire du registre général 15.

Cette requête est souvent utilisée en conjonction avec la variable spéciale &RETCODE qui contient la valeur du code de retour fourni par l'exécution de la dernière commande CMS lancée.

Exemple : &EXIT &RETCODE On sort de la macro-commande en cours d'interprétation avec le code de retour de la dernière commande CMS.

&GOTO

Objet : cette requête permet d'avoir des ruptures de séquences dans l'exécution de la macro-commande.

Format

```
-----------------------------------------
I                                       I
I    &GOTO      ( TOP )                 I
I               ( numéro-de-ligne )     I
I               ( étiquette )           I
I                                       I
-----------------------------------------
```

TOP il y a transfert à la première ligne de la macro-commande.

numéro-de-ligne l'instruction suivante est celle dont le numéro d'ordre est égal au numéro-de-ligne spécifié.

étiquette il y a branchement à la ligne commençant par l'étiquette indiquée. Une étiquette est une suite de caractères dont la longueur maximum est 11 et dont le prémier caractère est -.

Utilisation :

Cette requête est utilisée pour ne pas avoir un déroulement séquentiel de la macro-commande. Notons que toute instruction exécutable (commande CMS ou requête EXEC) peut commencer par une étiquette.

&IF

Objet : cette requête fournit à l'utilisateur la possibilité d'avoir une exécution conditionnelle d'une instruction.

Format :

```
-----------------------------------------------
I  &IF  (mot1) ( EQ NE ) (mot2) instruction  I
I       ( &* ) ( GT GE ) ( &* )              I
I       ( &$ ) ( LT LE ) ( &$ )              I
I                                            I
-----------------------------------------------
```

&* a le sens de tous les arguments arg1, arg2, ..., argN.

&$ a le sens de l'un des arguments arg1, arg2, ,....., argN.

Les opérateurs ont pour sens :

EQ égal

NE différent

GT plus grand que

GE supérieur ou égal à

LT inférieur à

LE inférieur ou égal à

Utilisation :

L'interpréteur commence par déterminer si la condition est vraie ou fausse. Si la condition est vraie il y a alors interprétation de l' "instruction" indiquée dans la requête. Sinon on

passe à la commande suivante.

La comparaison est numérique si les deux opérandes sont numériques; alphabétique s'ils sont alphabétiques. Dans tous les autres cas il y a une erreur de conversion.

Dans le cas où les deux opérandes sont alphabétiques, la longueur sur laquelle porte la comparaison est celle de la chaîne la plus longue. La chaîne la plus courte est complétée à droite par des blancs. La comparaison se fait selon l'ordre alphabétique, caractère par caractère et de gauche à droite.

Ainsi au test : DUPONT GT DUPOND il sera répondu oui.

L'instruction peut être toute commande CMS ou toute requête EXEC.

Exemple : &IF &A GT 275 &A=0

Si la variable &A est supérieure à 275, on la remet à zéro.

&LOOP

Objet : cette requête permet d'exécuter plusieurs fois une même séquence d'instructions.

Format :

```
-----------------------------------------
I                                       I
I   &LOOP  ( n )        ( m )           I
I          (étiquette) (condition)      I
I                                       I
-----------------------------------------
```

Utilisation :

Le bloc de commandes à exécuter est constitué des n lignes suivantes ou commence à la ligne suivante pour se terminer à la ligne commençant par l'étiquette. Cette dernière ligne est incluse dans le bloc.

La séquence de commandes ainsi spécifiée est exécutée m fois ou jusqu'à ce que la condition soit satisfaite.

Il n'y a pas de valeur par défaut pour ces deux paramètres.

La condition est testée avant l'exécution de chaque séquence. Lorsque l'on sort de la boucle la prochaine instruction est celle qui suit la dernière instruction du corps de boucle.

On peut avoir jusqu'à quatre niveaux d'imbrication de boucles.

Réponse : MAX DEPTH OF LOOP NESTING EXCEEDED

Dans ce cas le niveau d'imbrication est supérieur à quatre.

CONVERSION ERROR

Les paramètres n ou m ne sont pas des valeurs numériques.

EOF FOUND IN LOOP

La portée de la boucle dépasse la dernière instruction de la macro-commande.

LABEL NOT FOUND

L'étiquette donnée en paramètre n'existe pas.

Exemple :

```
&I=1
&LOOP -FIN &I GT 5
ASSEMBLE DMSINT.&I ( NOTERM
&IF &RETCODE NE 0 &EXIT &RETCODE
-FIN &I=&I+1
```

&PUNCH

Objet : cette requête envoie une ligne sur le perforateur de la machine virtuelle.

Format :

```
-----------------------------------------
I                                       I
I     &PUNCH      ligne                 I
I                                       I
-----------------------------------------
```

Utilisation :

On applique le mécanisme de substitution aux différents mots qui composent la ligne. Cette ligne est alors tronquée à partir du quatre-vingtième caractère ou complétée par des blancs à droite. Elle est ensuite envoyée sur le perforateur de la machine virtuelle.

Pour perforer plusieurs lignes sans appliquer le mécanisme de substitution, la requête employée sera &BEGPUNCH.

&READ

Objet : cette requête permet de lire une ou plusieurs instructions, des arguments ou des variables qui seront alors interprétées ou affectées.

Format :

```
-----------------------------------------
I                                       I
I    &READ     ( n )                    I
I              ( 1 )                    I
I              ( ARGS )                 I
I              ( VARS var1 var2 .... )  I
I                                       I
-----------------------------------------
```

n les n prochaines lignes traitées par l'interpréteur seront lues à la console. La valeur par défaut de n est 1.

ARGS on lit une ligne à la console et les mots de cette ligne sont affectés aux arguments comme dans la commande &ARGS.

VARS var1 var2 ..... les mots de la ligne lue sont affectés aux variables var1, var2, etc...

Utilisation :

Avec l'option n, les n lignes lues au terminal sont traitées comme si elles appartenaient a la macro-commande en cours d'interprétation. C'est à dire que ces lignes peuvent être aussi bien des commandes CMS que des requêtes de contrôle EXEC, des instructions

d'affectation ou même une ligne commentaire ou nulle. Remarquons que si une ligne lue est une requête &GOTO ou &LOOP, on annule le nombre de lignes restant à lire à la console. Si une ligne lue est elle-même une requête &READ on ajoute le nombre de lignes à lire au nombre restant.

Avec l'option ARGS, on redéfinit les arguments &1, &2, ..., &N. La variable spéciale &INDEX est remise à jour. Comme lors de l'appel d'une macro-commande ou de l'exécution de la requête &ARGS, le symbole % permet d'ignorer un argument.

Avec l'option VARS, les mots de la ligne lue sont affectés aux variables var1, var2, etc.... Ces variables sont analysées de la même façon que lorsqu'il s'agit d'une affectation. C'est à dire que le mécanisme de substitution s'arrête à l'avant dernier caractère du mot représentant la variable.

Exemple : &READ VARS &A &V&I &V23

les mots de la ligne lue seront affectés aux variables &A, &V12 (si &I vaut 12), &V23.

&SKIP

Objet : nous pouvons par cette requête sauter un certain nombre de lignes à l'intérieur de la macro-commande.

Format :

```
-----------------------------------------
I                                       I
I      &SKIP     ( n )                   I
I                ( 1 )                   I
I                                       I
-----------------------------------------
```

Utilisation :

La requête &SKIP fait transférer le contrôle après les n lignes suivantes. Le paramètre n peut être positif ou négatif. Le nombre de lignes sautées est égal à la valeur absolue de n. Si n est égal à 0 on passe à la ligne suivante. Dans le cas où n n'est pas spécifié la valeur prise par défaut est 1.

Réponse : CONVERSION ERROR DURING &SKIP OR &GOTO

ceci indique que la valeur n n'est pas de type arithmétique.

SKIP BEFORE THE FIRST LINE

Dans ce cas n est négatif et sa valeur absolue est supérieure au numéro de la ligne courante moins un. Ce qui équivaut à un transfert avant la ligne 1.

Exemple : &SKIP 3

On saute les 3 prochaines lignes.

&SPACE

Objet : cette requête permet d'imprimer des lignes blanches sur la console de l'utilisateur.

Format :

```
-----------------------------------------
I                                       I
I     &SPACE      ( n )                 I
I                 ( 1 )                 I
I                                       I
-----------------------------------------
```

Utilisation :

n indique le nombre de lignes blanches qui seront imprimées à la console. Si n n'est pas spécifié, la valeur prise par défaut sera 1.

Réponse : CONVERSION ERROR

La valeur n n'est pas de type arithmétique.

&STACK

Objet : cette requête a pour objet de mettre dans la pile associée à la console de l'utilisateur une ligne dont les mots seront passés par le mécanisme de substitution.

Format :

```
-----------------------------------------
I                                       I
I   &STACK   ( LIFO )    ligne          I
I            ( FIFO )                   I
I                                       I
-----------------------------------------
```

Utilisation :

Les éléments qui constituent la ligne de cette requête subissent d'abord le mécanisme de substitution. La ligne ainsi constituée, qui peut être éventuellement nulle (à ne pas confondre avec une ligne blanche) est rangée dans la pile associée à la console de l'utilisateur.

FIFO (first-in, first-out) la ligne est rangée suivant l'ordre : premier arrivé, premier servi.

LIFO (last-in, first-out) l'ordre suivant lequel la ligne est rangée est : dernier arrivé, premier servi.

Lorsque l'option n'est pas spécifiée, la valeur prise par défaut est FIFO. La ligne est

rangée dans la forme qu'elle a après substitution. Pour ranger une ou plusieurs ligne (y compris une ligne blanche) il faut utiliser la requête &BEGSTACK décrite précédemment.

Note : le caractère de fin de ligne logique (≠) n'est pas analysé par les processus de CMS. En conséquence, si on veut l'utiliser, c'est à dire mettre sur la même ligne physique plusieurs lignes logiques qui seront substituées et empilées, on procédera de la façon suivante :

- sous EDIT, utiliser la commande SETCHAR ou ALTER pour donner à un caractère (≠ ou un autre), la valeur héxadécimale 15. Cette valeur représente le caractère de fin de ligne logique de façon interne. Le format de SETCHAR est :

  SETCHAR, ≠ 21

  (le code 15 doit être mis en décimal).

- changer le caractère de fin de ligne logique par la commande :

  CP TERM LINEND OFF

&TIME

Objet : cette requête détermine les informations sur le temps CPU consommé par les commandes CMS, qui doivent être imprimées sur la console de l'utilisateur.

Format :

```
-----------------------------------------
I                                       I
I   &TIME     ( ON )                    I
I             ( OFF )                   I
I             ( RESET )                 I
I             ( TYPE )                  I
I                                       I
-----------------------------------------
```

ON les temps CPU (virtuel et total) sont réinitialisés avant l'exécution de chaque commande CMS. Ils sont imprimés à la fin de l'exécution de la commande.

OFF les temps ne sont ni réinitialisés ni imprimés.

RESET les temps CPU sont réinitialisés au moment de l'interprétation de cette requête.

TYPE les temps CPU sont imprimés au moment de l'interprétation de cette requête puis réinitialisés.

Utilisation :

La forme de l'information imprimée à la console de l'utilisateur est :

T=X.XX/Y.YY HH:MM:SS où

X.XX est le temps CPU consommé par la machine virtuelle depuis le dernière initialisation. Il est exprimé en secondes et centièmes de seconde.

Y.YY est le temps total (temps virtuel plus le temps de simulation de la machine virtuelle) consommé depuis sa dernière initialisation. Il est exprimé en secondes et centièmes de seconde.

HH:MM:SS est l'heure du jour.

Les temps CPU (virtuel et total) sont initialisés avant l'exécution de la première instruction de la macro-commande.

Réponse : T=0.75/1.27 10:52:43

il y a eu 75 centièmes de seconde de consommés par la machine virtuelle et 1 seconde 27 centièmes de temps total depuis la dernière initialisation.

&TYPE

Objet : cette requête permet d'imprimer à la console de l'utilisateur une ligne qui est passée par le mécanisme de substitution.

```
-----------------------------------------
I                                       I
I     &TYPE     ligne                   I
I                                       I
-----------------------------------------
```

Utilisation :

Les mots de la ligne passent par le mécanisme de substitution. Puis la ligne est imprimée sur le terminal de l'utilisateur.

Pour imprimer plusieurs lignes sans les faire passer par le mécanisme de substitution, on utilisera la requête &BEGTYPE.

5 FONCTIONS INTERNES

Le langage de contrôle fournit la possibilité d'utiliser trois fonctions internes. Une fonction interne est constituée d'un mot clé suivi d'une liste d'arguments. Les fonctions internes ne peuvent être utilisées que lorsqu'elles apparaissent en partie droite d'une affectation.

5.1 &DATATYPE

Cette fonction permet de savoir si un élément est de type numérique ou alphabétique. Son format est :

&DATATYPE (mot)

La valeur du résultat fourni par l'évaluation de cette fonction est NUM ou CHAR selon le type du mot fourni en zone argument.

Exemple : &A='ABCD'

&B=&DATATYPE (&A)

&TYPE &B

A la console s'imprimera le résultat CHAR.

5.2 &LENGTH

Cette fonction permet de connaître la longueur, en termes de caractères, d'un mot fourni en zone argument. Le format est :

&LENGTH (mot)

On ne peut utiliser cette fonction qu'en partie droite d'une affectation. Si le mot est de type numérique, on obtient la longueur de sa représentation alphabétique, y compris le signe s'il est négatif.

Exemple : &A=100-250

&B=&LENGTH (&A)

&TYPE &B

On aura la valeur 4 qui sera imprimée à la console.

5.3 &SUBSTR

Cette fonction a pour objet d'extraire d'une chaîne de caractères une sous-chaîne spécifiée par deux arguments. Le format de cette fonction est :

&SUBSTR mot(i:j)

La valeur retournée par cette fonction est la sous-chaîne extraite du mot fourni, qui commence au i ième caractère et dont la longueur est j.

Exemple : &A=ABCDE

&B=&SUBSTR &A(2:3)

&TYPE &B

La valeur de &B qui sera imprimée à la console est la sous-chaîne BCD.

6 VARIABLES SPECIALES

Une des originalites du langage de contrôle est de fournir à l'utilisateur la possibilité d'obtenir des renseignements sur le contexte dans lequel est utilisée la macro-commande. Ces variables ont un nom qui est un mot réservé. L'utilisateur ne peut que connaître leur contenu et donc ne peut pas les modifier. On distingue trois sortes de variables suivant le type de renseignements fournis :

- les variables relatives à la façon dont se déroule l'interprétation de la macro-commande.
- les variables relatives à l'état de la machine virtuelle.
- les variables d'intérêt général.

6.1 VARIABLES SUR L'ETAT DE LA MACRO-COMMANDE

&INDEX

Cette variable est de type numérique et contient le nombre d'arguments qui ont été passés au moment de l'appel, ou qui ont été redéfinis par les requêtes

&ARGS ou &READ ARGS.

&LINENUM

C'est une variable de type numérique qui contient le numéro d'ordre de la ligne de la macro-commande en cours d'interprétation.

&RETCODE

Cette variable est de type numérique et contient le code de retour de la dernière commande CMS qui a été exécutée.

6.2 VARIABLES SUR L'ETAT DE LA MACHINE VIRTUELLE

&TYPEFLAG

Cette variable est de type alphabétique et contient la valeur HT ou RT. La valeur HT (halt typing) indique que les impressions console ont été inhibées. La valeur RT (ready typing) spécifie que les impressions console sont autorisées.

&READFLAG

Cette variable est de type alphabétique et contient la valeur CONSOLE ou STACK. Cette valeur dépend de la provenance de la prochaine ligne qu'un processus aura essayé de lire à la console. En d'autre

termes, on aura la valeur STACK si la pile associée à la console de la machine virtuelle n'est pas vide; sinon on aura la valeur CONSOLE.

&USERID

C'est une variable de type caractère qui contient le nom de la machine virtuelle. Ce nom est celui qui est fourni en paramètre lors de l'exécution de la commande LOGIN.

&CORE

C'est une variable de type numérique. Elle contient la taille de mémoire libre dont dispose l'utilisateur à un instant donné. L'unité dans laquelle est exprimée cette variable est le K-octets (1024 octets de 8 bits chacun).

&DISK

C'est une variable de type numérique. Elle contient le nombre d'enregistrements libres dont l'utilisateur dispose sur son disque principal (A-disque). La taille d'un enregistrement est le taille standard adoptée par le système CMS; c'est à dire qu'il a une longueur de 800 octets.

6.3 VARIABLES GENERALES

&DATE

La valeur contenue dans cette variable a le format MM/JJ/AA, où MM est le mois, JJ le jour et AA l'année.

&HOUR

Cette variable donne l'heure du jour. Son format est : HH:MM:SS qui donne une représentation de l'heure en heure, minute et seconde.

7 DESCRIPTION DES MESSAGES D'ERREUR

Lorsque l'interpréteur rencontre un erreur dans une requête de contrôle EXEC, il y a impression d'un message d'erreur à la console de l'utilisateur (à moins que celui-ci ait inhibé toute sortie). La formé du message est :

ERROR IN EXEC FILE filename, LINE nn, description de l'erreur

dans lequel :

- filename : est le nom de lacro-commande EXEC.
- nn : est le numéro de la ligne où s'est produite l'erreur.
- description de l'erreur : est l'une des conditions

décrites ci-dessous. Le code de retour indiqué est passé au programme appellant selon les conventions de CMS (par l'intermédiaire du registre 15).

801 AN &BEG... BLOCK IS NOT CLOSED

Au cours de l'interprétation d'une requête &BEGPUNCH, &BEGSTACK ou &BEGTYPE on a trouvé une fin de fichier avant &END.

802 CONVERSION ERROR DURING &SKIP OR &GOTO

Le paramètre donné avec ces requêtes n'est pas de type numérique.

803 A STRING IS NOT CLOSED

Il manque une apostrophe dans une chaîne.

804 TOO MANY ARGS

Plus de 30 arguments sont passés en paramètre à la macro-commande.

805 MAX DEPTH OF LOOP NESTING EXCEEDED

Le nombre d'imbrication de boucles est supérieur à 4.

806 PERMANENT DISK ERROR

Lecture disque impossible. Consulter le responsable de l'installation.

807 INVALID SYNTAX

Un mot d'une requête n'a pas une syntaxe correcte.

808 INVALID FORM OF CONDITION OPERATOR

L'opérateur d'une comparaison n'est pas EQ, NE, LE, LT, GE ou GT.

809 QUOTE MISSING IN AN ARGUMENT

Un paramètre d'appel contient une chaîne de caractères dans laquelle il manque une apostrophe.

810 MISUSE OF SPECIAL VARIABLE

Il y a une affectation à une variable de même nom qu'une variable spéciale.

811 ERROR IN &ERROR ACTION

L' "action" de la requête &ERROR est une commande CMS qui provoque elle-même une erreur.

812 CONVERSION ERROR

Un élément devant être de type numérique ne l'est pas.

813 INVALID FORM OF CONDITION IN A LOOP

La syntaxe d'une condition exprimée dans une requête &LOOP est incorrecte.

815 EOF FOUND IN LOOP

La portée d'une boucle dépasse la fin de la macro-commande.

816 INVALID CONTROL WORD

Le premier mot d'une ligne, après substitution, n'est pas un mot réservé.

817 A STRING IS FOUND IN A NUMERIC OPERATION

Un élément d'une expression arithmétique est de type caractère.

818 MISUSE OF BUILT-IN &SUBSTR

La syntaxe de la fonction spéciale &SUBSTR n'est pas respectée.

819 LABEL NOT FOUND

On ne trouve pas l'étiquette spécifiée dans une requête &GOTO ou &LOOP.

820 AFFECTATION ERROR IN &READ VARS

Erreur au cours de l'affectation à une variable.

821 SKIP BEFORE THE FIRST LINE

Tentative de branchement avant la première ligne de la macro-commande.

822 CONVERSION ERROR IN CONDITION TEST

L'un des opérandes d'une condition est de type caractère.

823 STRING NOT CLOSED IN AN ELEMENT OF COMPARISON

Un opérande d'une comparaison contient une chaîne de caractères dans laquelle il manque une apostrophe.

824 ERROR IN VALUE OF PARAMETERS OF BUILT-IN &SUBSTR

L'un des paramètres de la fonction ne correspond aux bornes de la chaîne spécifiée.

825 ARITHMETIC OPERATOR NOT FOUND

Il manque un opérateur dans une expression arithmétique.

C H A P I T R E I I I

D E S C R I P T I O N I N T E R N E
E T L O G I Q U E
D E E X E C +

1 PRESENTATION

Le macro-processeur se compose de deux parties. D'une part on trouve une partie qui est résidente dans le noyau du système CMS. La seconde partie est un module chargeable dans la mémoire libre dont dispose à un instant donné le système.

Le module résidant dans le noyau du système a essentiellement pour but d'amorcer le processus d'interprétation. Il acquiert une zone de mémoire libre dans laquelle il demandera que soit chargé le second module. C'est ce second module qui est en fait réellement l'interpréteur.

Nous donnerons tout d'abord une description succincte du module résidant dans le noyau. Par contre nous nous attacherons principalement dans ce chapitre à décrire l'implémentation que nous avons faite pour l'interpréteur. Plus particulièrement, nous donnerons une description de la partie de ce module qui fait une analyse globale de la macro-commande à interpréter. Nous décrirons ensuite le mécanisme de substitution de façon interne. Enfin nous analyserons le fonctionnement de la partie interprétation des commandes proprement dite.

2 DESCRIPTION INTERNE DE DMSEXC

Ce module prend le contrôle après que DMSINT (qui est le premier module activé après l'émission d'une commande à la console) ait reconnu que la commande à traiter était une macro-commande EXEC. Les conventions de liaison adoptées pour ce transfert de contrôle sont les conventions standards du système CMS. C'est à dire que le registre général 1 référence une liste d'éléments qui représente la commande mise sous la forme d'une suite de mots de 8 caractères chacun.

Après avoir incrémenté le niveau de récursion, s'il est égal à un, il y a demande d'une zone de mémoire libre. Cette zone ne doit se trouver ni dans la zone libre utilisable par des programmes utilisateur, ni dans la zone "transient" réservée au système. L'interprétation de la macro-commande peut en effet conduire à l'exécution de commandes qui, pour s'exécuter, acquièrent ces zones. Ceci aurait pour effet d'écraser l'interpréteur. La zone acquise doit donc se trouver dans la partie réservée aux modules du système. Si le chargement de l'interpréteur a pu se dérouler correctement, le contrôle lui est alors passé.

Au retour, il y aura décrémentation du niveau de récursion et restitution à la mémoire libre de la zone occupée par l'interpréteur.

3 L'INTERPRETEUR DE COMMANDES

3.1 UTILISATION DU LANGAGE D'ECRITURE

Le langage choisi pour développer l'interpréteur EXEC+ est le langage PL/S II (Programming Language System). C'est un langage de haut niveau, dérivé du langage PL/I, dans lequel on retrouve le même genre d'instructions comme DO, IF, ELSE, END, GOTO ainsi que les instructions d'appel et de retour CALL et RETURN. Ce langage permet aussi la manipulation de données déclarées BASED ; c'est à dire dont l'allocation est laissée aux soins de l'utilisateur. La différence essentielle entre ce type de langage et les langages de haut niveau classique est la possibilité d'une part de pouvoir insérer directement du code en langage d'assemblage, d'autre part de pouvoir allouer des registres généraux. Pour insérer du code en langage d'assemblage on utilise l'instruction GENERATE. Pour allouer de façon temporaire ou permanente des registres généraux on utilise l'instruction RESPECIFY avec les options RESTRICTED ou UNRESTRICTED.

Le produit du compilateur est un programme en langage d'assemblage composé d'une seule section de contrôle. La génération de ce programme objet peut être contrôlée par le programmeur qui fera usage d'options spécifiées dans l'en-tête de procédure. On peut ainsi spécifier par exemple

la réentrance d'un programme par l'option REENTRANT ou les conventions de liaison entre les diverses procédures. Sauf indication contraire, les conventions de liaisons employées sont celles utilisées par le système OS/360. Rappelons en brièvement le principe : à chaque section de programme est associée une zone de sauvegarde ("save area") dans laquelle seront rangés les registres généraux au moment d'un appel. Les différentes zones de sauvegarde sont chainées entre elles aussi bien dans le sens appelé-appelant qu'inversement. L'adresse de la zone du programme courant est contenue dans le registre 13. Si une section est réentrante sa zone est acquise dynamiquement. Notons à ce propos qu'il convient d'examiner attentivement le code généré par le compilateur pour la réalisation de ces opérations d'acquisition et de restitution de mémoire libre. En effet, pour les mêmes raisons que celles énoncées dans le paragraphe 2, les acquisitions de mémoire pour ces zones ne doivent pas se faire ni dans la partie "transient" ni dans la mémoire libre de l'utilisateur.

La réalisation que nous avons faite pour le macro-processeur EXEC+ se compose de deux modules principaux. Le premier EXECTOR, a pour objet la partie acquisition d'une commande et analyse. Le second, STARTCOM, est quant à lui, chargé de l'interprétation s'il s'agit d'une requête du langage de contrôle ; ou du lancement de la commande si c'est une commande du langage de commande.

3.2 EXECTOR : ANALYSE D'UNE MACRO-COMMANDE

Ce composant est un des éléments essentiels dans l'organisation interne du macro-processeur. Son rôle est triple :

- d'une part c'est lui qui reçoit le contrôle des modules demandant l'exécution d'une macro-commande. Il assumera donc de ce fait les communications entre ces modules et ceux de l'interpréteur ; il sera d'autre part chargé de gérer globalement les variables de l'interpréteur.
- deuxièmement, il assure la partie acquisition d'une commande. Celle-ci pourra provenir soit directement de l'utilisateur, soit d'un fichier de type EXEC.
- enfin, il effectue une analyse globale de la ligne afin de discerner les mots et les modifiera en faisant les substitutions nécessaires.

3.2.1 GESTION GLOBALE DES VARIABLES DE L'INTERPRETEUR

Il y a deux façons possibles de demander l'exécution d'une macro-commande EXEC. Soit cette demande provient directement de l'utilisateur qui l'a rentrée par l'intermédiaire de la console. Soit elle provient de

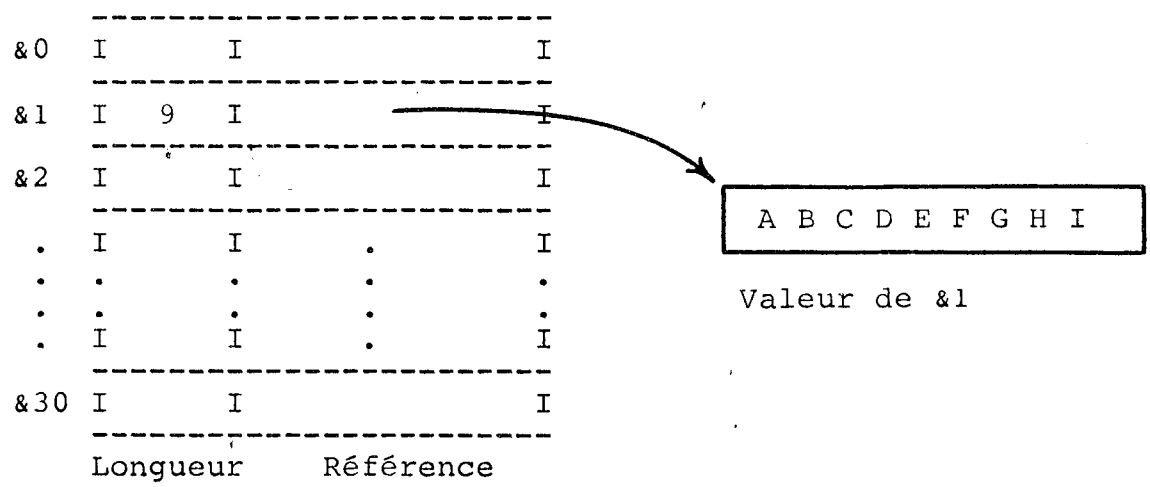
l'interpréteur lui-même qui, parmi les commandes CMS, a détecté une commande EXEC. Dans le premier cas, selon un schéma classique propre au système CMS, la commande a d'abord été mise sous forme d'une liste d'éléments. Pour cela, le processus de CMS chargé de cette opération sélectionne dans la ligne reçue du terminal une chaîne comprise entre deux blancs et la tronque après le huitième caractère. Comme dans notre prototype les arguments ont une longueur quelconque et que nous avons introduit les chaînes de caractères, il convient donc tout d'abord de récupérer la commande sous sa forme initiale au lieu de sa forme "atomisée".

Disposant à présent d'une commande EXEC, la première étape consiste à déterminer s'il existe un fichier de type EXEC dont le nom est le même que celui de la commande émise. Pour cela, on utilise un module existant de CMS (DMSSTATE) auquel on fournit en paramètre le nom du fichier et son type. Ce module recherche alors sur les divers disques (au sens CMS) auquel l'utilisateur a accès si le fichier indiqué existe. Cette recherche est faite selon un ordre prédéfini qui est A, B, ..., S où A est le disque principal utilisateur et S le disque système. Si la recherche est infructueuse, le macro-interpréteur redonne le contrôle au module qui l'avait appelé avec un code de retour approprié.

Par contre, dans le cas où cette recherche s'est soldée

par un succès, on acquiert une zone de mémoire libre. Dans celle-ci seront rangées toutes les variables propres à l'interpréteur qui à un instant donné définissent l'état d'interprétation de la macro-commande. Il s'agit ensuite d'initialiser un certain nombre de ces variables ; notamment celles qui représentent les valeurs par défaut de certaines options des requêtes du langage de contrôle.

La phase suivante est le traitement des paramètres passés lors de l'appel de la macro-commande. Il s'agit d'affecter ces différentes valeurs aux variables spéciales &1, &2, etc... La forme même du nom de ces arguments fait qu'il suffit de référencer ces valeurs par un tableau à 31 entrées. Le nom de l'argument servira d'index dans ce tableau.



Représentation des Arguments

Figure 1

Cette phase peut conduire à un arrêt du mécanisme d'interprétation de deux façons. Soit parce que la syntaxe propre aux arguments n'a pas été respectée. Soit parce que le nombre d'arguments passés a été supérieur à la limite autorisée qui est de 30. L'affectation des arguments se fait par position. Le caractère de séparation entre deux paramètres est le caractère blanc, sauf s'il se trouve à l'intérieur d'une chaîne de caractères comprise entre deux apostrophes.

Après cette première phase qui peut être considérée comme celle de l'initialisation de l'état d'interprétation de la macro-commande, commence l'interprétation proprement dite des lignes de la macro-commande.

3.2.2 CHOIX DE LA COMMANDE

Le premier cas qu'il convient d'envisager dans le choix de la commande courante, est celui qui consiste à déterminer si la commande précédente était une commande CMS dont l'exécution a provoqué une erreur. Dans ce cas, la prochaine commande sera celle qui correspond à la condition "ON-ERROR" propre à la macro-commande. C'est à dire que la commande sera l' "action" de la requête &ERROR si elle a été utilisée ou sinon, on prendra l'option par défaut qui est &CONTINUE. Néanmoins, on commencera tout d'abord par vérifier que la

commande CMS qui a provoqué l'erreur n'est pas elle même l' "action" correspondant au "ON-ERROR". Dans ce cas, il faudra arrêter l'interprétation de la macro-commande qui sinon bouclerait. Dans le cas contraire, la commande courante sera l' "action" définie par le requête &ERROR.

Si la commande précédente n'était pas une commande CMS ayant provoqué une erreur, il y a deux façons de choisir la commande courante. On examine pour cela le quantum de lignes restant à lire à la console.

Dans le premier cas, s'il est positif, cela signifie qu'il y a eu précédemment interprétation d'une requête &READ N et que le nombre de lignes lues est inférieur à N. Dans ce cas, on demande, par l'intermédiaire d'un module CMS, une lecture console. La chaîne de caractères lue devient alors la commande courante à interpréter.

Dans le second cas, si le quantum est nul, la prochaine commande sera l'enregistrement suivant du fichier de type EXEC qui constitue physiquement la macro-commande. Nous devons cependant analyser au préalable certaines variables d'état de la macro-commande. Celles-ci peuvent en effet indiquer que l'on est en train d'interpréter des commandes qui se trouvent dans le corps d'une boucle. Dans cette hypothèse, trois choix sont possibles en ce qui concerne la commande suivante.

- c'est la première commande du corps de boucle.
- la prochaine commande appartient à la boucle mais ce n'est pas la première.
- la commande suit la dernière commande de la boucle.

Nous allons examiner chacun de ces trois cas.

a) Dans le cas où la commande suivante est la première du corps de boucle, on doit consulter certaines variables d'état afin de déterminer s'il y a lieu ou non d'entamer un tour de boucle.

Soit la boucle est "par condition". C'est à dire que l'en-tête de boucle est

&LOOP .... condition

On commence alors par déterminer la valeur de la condition. Si la condition est VRAIE, on ne doit pas exécuter la boucle. On décrémente le niveau de boucle et la commande suivante est celle qui suit la dernière commande du corps de boucle. Puis on vérifiera s'il y a imbrication de boucle. Dans ce cas, on doit réappliquer les mêmes investigations pour cette nouvelle commande. Si la condition testée est FAUX, la première commande du corps de boucle devient la commande courante.

Si la boucle est à nombre de tour fixé, c'est à dire que l'en-tête de boucle est

&LOOP ..... M

on vérifie que ce nombre est positif. S'il l'est, on le

décrémente et la commande courante est la première du corps de boucle. S'il est nul, on décrémente le niveau de boucle puis on prend comme commande courante celle qui suit la dernière de la boucle. Et là encore, on vérifiera si cette nouvelle commande appartient à une boucle.

b) Si la commande appartient à une boucle mais n'est pas la première, elle devient la commande courante.

c) Si la commande choisie est celle qui suit la dernière commande du corps de boucle, on prend comme nouvelle commande la première de la boucle ce qui nous ramène au cas a) ci-dessus.

La prochaine commande à interpréter ayant été déterminée et amenée en mémoire la phase suivante va consister à l'analyser et à effectuer les substitutions nécessaires.

3.2.3 ANALYSE ET SUBSTITUTION DANS LA COMMANDE

La première analyse à effectuer, consiste à déterminer si la commande est constituée d'une ligne vide ou si c'est une ligne commentaire. Dans ce cas on continue l'interprétation en revenant à la phase d'acquisition d'une commande.

Sinon on examine une variable d'état qui indique si la commande courante appartient à un groupe &BEG... Dans ce cas, il n'y a pas d'analyse ni de substitution à faire pour cette ligne qui n'est en fait qu'une donnée pour la requête &BEG... Elle est alors interprétée (empilée, perforée ou imprimée) directement.

L'analyse d'une commande est un processus de reconnaissance. Il s'agit, ayant d'une part une chaîne de caractères et d'autre part une liste de séparateurs, de localiser les différents mots de la commande. L'analyse de la ligne se fait de gauche à droite. Un mot est toute chaîne de caractères comprise entre deux délimiteurs. Un délimiteur est un blanc ou l'un des caractères suivants :

+ - = ( ) :

On notera que tout délimiteur perd sa valeur de délimiteur s'il appartient à un élément de type chaîne de caractères. C'est à dire s'il est compris entre deux apostrophes. Pour considérer l'apostrophe elle-même comme un caractère banal, il faut la doubler.

L'analyse de la commande étant faite, la phase suivante consiste en l'application du mécanisme de substitution des variables à l'intérieur des mots. Ce mécanisme, dont une description détaillée est faite au paragraphe suivant, s'applique à tous les mots d'une ligne. Il convient

néanmoins de signaler quelques cas particuliers.

Tout d'abord, si le mot est un mot réservé du langage de commande (&ARGS, &GOTO, ...) ce mécanisme n'est pas appliqué. D'autre part les mots qui constituent la partie "condition" de certaines requête de contrôle ne passent pas, dans cette phase, par ce mécanisme. La substitution pour ces mots n'est faite qu'au moment de l'interprétation afin de pouvoir tenir compte du cas particulier où un mot (opérande) est substitué par une chaîne nulle. Enfin, si l'on reconnait que la commande est une affectation ou si l'on a la requête &READ VARS, on doit positionner une variable afin que le processus de substitution appliqué au nom d'une variable s'arrête à l'avant dernier caractère du mot représentant le nom.

La substitution du mot étant faite, on remplace alors dans la commande le mot par sa nouvelle valeur. Lorsque tous les mots sont passés par ce mécanisme, la commande se trouve dans sa forme définitive prête pour l'interprétation. Le contrôle est alors passé au module STARTCOM, décrit ultérieurement, qui est chargé de l'interprétation proprement dite des commandes.

Au retour, s'il n'y a pas eu d'erreur dans l'interprétation d'une requête de contrôle, on retourne en tête du processus, c'est à dire à la phase d'acquisition

d'une nouvelle commande. Dans le cas où il y a eu une erreur, il y a impression d'un message à la console de l'utilisateur, puis retour au processus qui a demandé l'exécution de cette macro-commande.

3.3 SUBSTIT : MECANISME DE SUBSTITUTION

Les éléments mis en jeu par le mécanisme de substitution sont : d'une part, un mot ; d'autre part, les arguments et les variables locales, globales et spéciales. Le processus peut se résumer comme étant le remplacement dans le mot concerné du nom d'une variable par sa valeur.

L'analyse du mot se fait caractère par caractère et de droite à gauche. Trois cas sont envisagés selon le caractère analysé.

Premièrement, le caractère est une apostrophe. On a donc dans le mot un élément de type chaîne de caractère. Aucune analyse ne sera donc faite pour les caractères qui se trouvent dans cet élément ; c'est à dire entre les deux apostrophes qui délimitent la chaîne. Cependant, on devra supprimer les deux délimiteurs de chaîne et transformer les apostrophes doubles de la chaîne en apostrophe unique.

Deuxièmement, le caractère analysé est un point. On est

en présence de l'opérateur de concaténation. Cette information est donc mémorisée pour qu'au moment d'une substitution les caractères à droite de cet opérateur ne fassent pas partie du nom d'une variable.

Enfin, le caractère est un &. On se trouve dans cas en présence d'une variable. Le nom de cette variable se compose de l'ensemble des caractères comprenant ce & et tous les caractères à sa droite, jusqu'au symbole de concaténation s'il existe ou sinon jusqu'au dernier caractère droit du mot. On détermine alors le type de cette variable (argument, variable locale, globale ou spéciale) et on remplace dans le mot le nom par la valeur.

Donnons sur un exemple le principe de fonctionnement de ce mécanisme. Soit la macro-commande :

```
&ARGS EXT INT LINE COND SCAN SUBS
&A=S
&I=1
&LOOP -FIN &INDEX
PRINT 'DM'&A.&&I ASSEMBLE
-FIN &I=&I+1
```

Cette macro-commande a pour objet d'imprimer les fichiers DMSEXT, DMSINT, ..., DMSSUBS de type ASSEMBLE. Intéressons nous au mécanisme de substitution du mot 'DM'&A.&&I de la cinquième commande. Au pas N de la boucle,

&I vaut N. Avec N=3 on aura par exemple pour les différentes étapes de ce mot :

'DM'&A.&&I

'DM'&A.&3 on remplace &I par 3

'DM'&A.LINE on remplace &3 par LINE

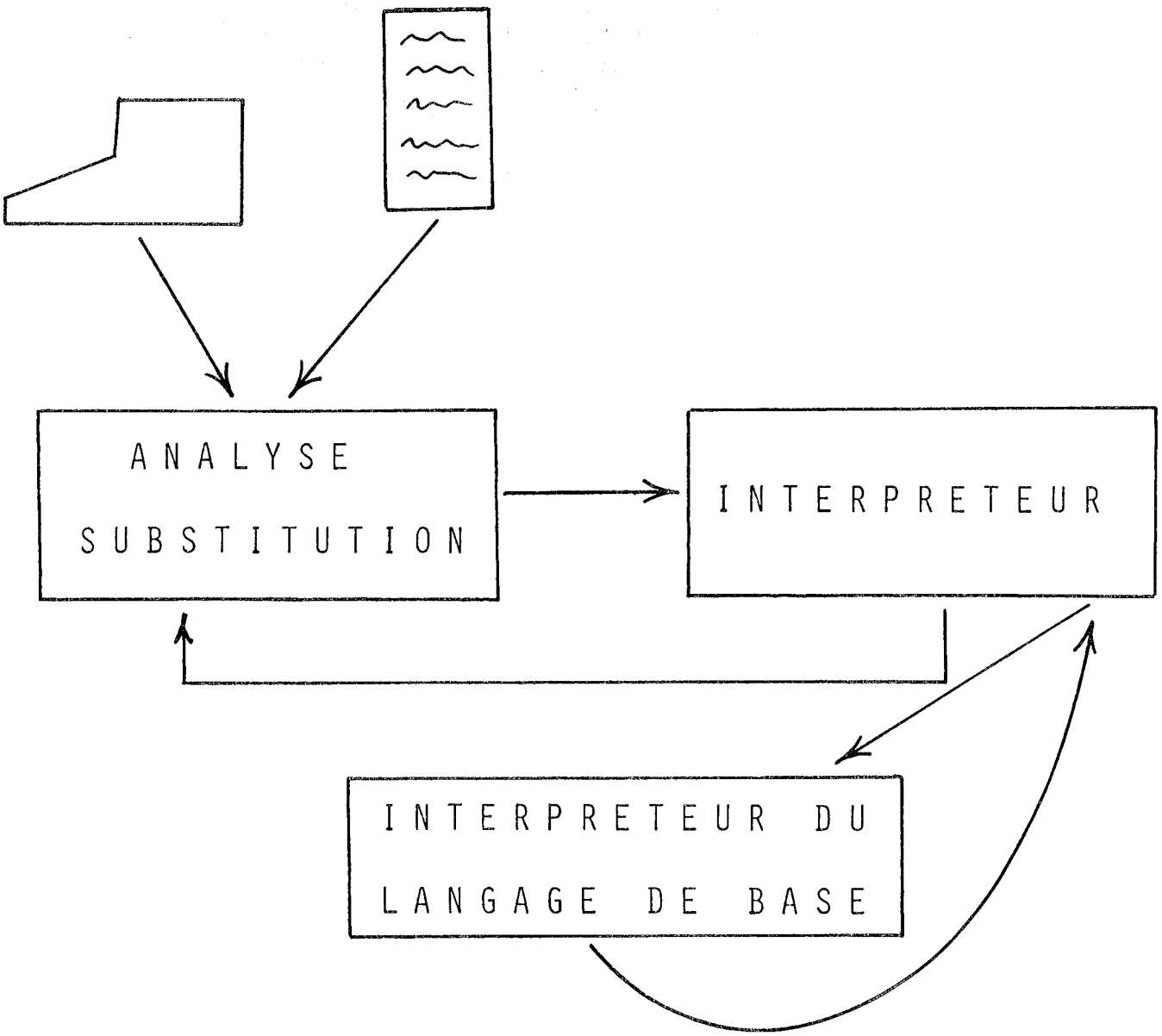
'DM'S.LINE on remplace &A par S

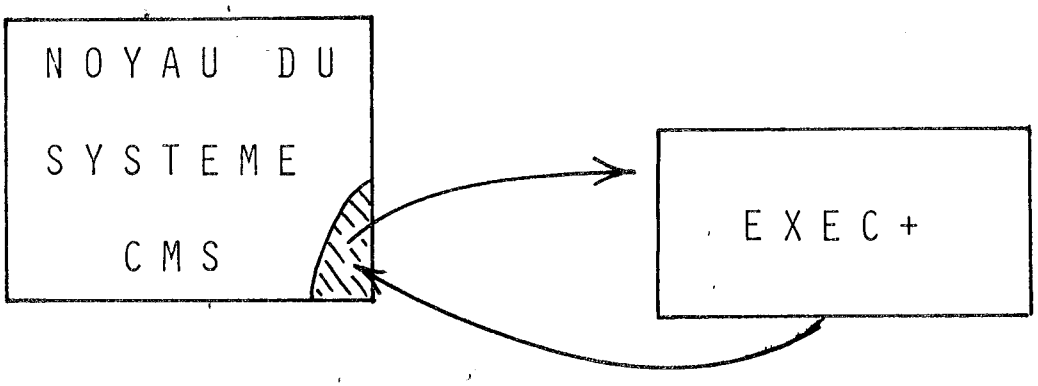
On a une concaténation, donc le nom de la variable à substituer est &A et non &A.LINE

DMS.LINE reconnaissance d'une chaîne

DMSLINE on effectue la concaténation

Notons que si la valeur d'une variable contient le caractère &, après substitution du nom de la variable par sa valeur, on ne réapplique pas le mécanisme de substitution pour la partie substituée. Ceci permet au processus de substitution de s'arrêter en un nombre fini de pas. Après substitution d'une variable dans le mot, l'analyse reprend à partir du caractère qui se trouve à gauche du caractère & qui a provoqué la substitution. On répète le processus jusqu'à épuisement des caractères.


ANALYSE
SUBSTITUTION
INTERPRETEUR
INTERPRETEUR DU
LANGAGE DE BASE


NOYAU DU
SYSTEME
CMS
EXEC+

3.4 STARTCOM : INTERPRETATION DES COMMANDES

Ce module est, avec le module EXECTOR précédemment décrit, l'élément essentiel dans l'organisation interne du macro-processeur. Comme nous l'avons vu, le premier module avait pour but de sélectionner une commande et de l'analyser afin de la transformer pour la mettre sous la forme dans laquelle elle sera interprétée. Ce module, quant à lui, est chargé de l'interprétation proprement dite des commandes. Il est appelé par EXECTOR qui lui passe comme paramètre d'entrée la commande sous sa forme définitive. Il interprète alors cette commande, c'est à dire qu'il entreprend les actions définies par les requêtes.

Tout d'abord, on détermine si la commande appartient à un groupe &BEGSTACK, &BEGPUNCH ou &BEGTYPE. Dans ce cas, la ligne est empilée, perforée ou imprimée selon le cas, puis il y a retour au module appelant.

Sinon, on analyse le premier caractère de la ligne. Si ce caractère est &, alors nous avons une requête de contrôle. Dans le cas contraire, c'est que l'on est en présence d'une commande du langage de commande.

3.4.1 INTERPRETATION DES REQUETES DE CONTROLE

Dans le cas d'une requête de contrôle, on commence par examiner la valeur d'une variable d'état, interne au macro-processeur, qui traduit l'option de contrôle choisie par l'utilisateur. Si cette valeur correspond à l'option "ALL" de la requête &CONTROL, on imprime à la console la requête courante.

Nous déterminons ensuite, par l'analyse du premier mot de la requête son type. Si ce mot est un mot réservé, on applique le traitement correspondant à cette requête dont nous donnerons une description pour les principales. Dans le cas contraire, c'est que l'on est en présence d'une affectation.

3.4.1.1 TRAITEMENT DES PRINCIPALES REQUETES

En ce qui concerne les requêtes &BEGSTACK, &BEGPUNCH, &BEGTYPE, &STACK, &PUNCH et &TYPE, on applique quelques modifications sur la ligne et on utilise pour chaque type de requête un module de CMS.

Pour la requête &ERROR, il suffit de ranger dans la zone de données réservées à l'interprétation de la macro-commande la partie "action" de la requête dans sa

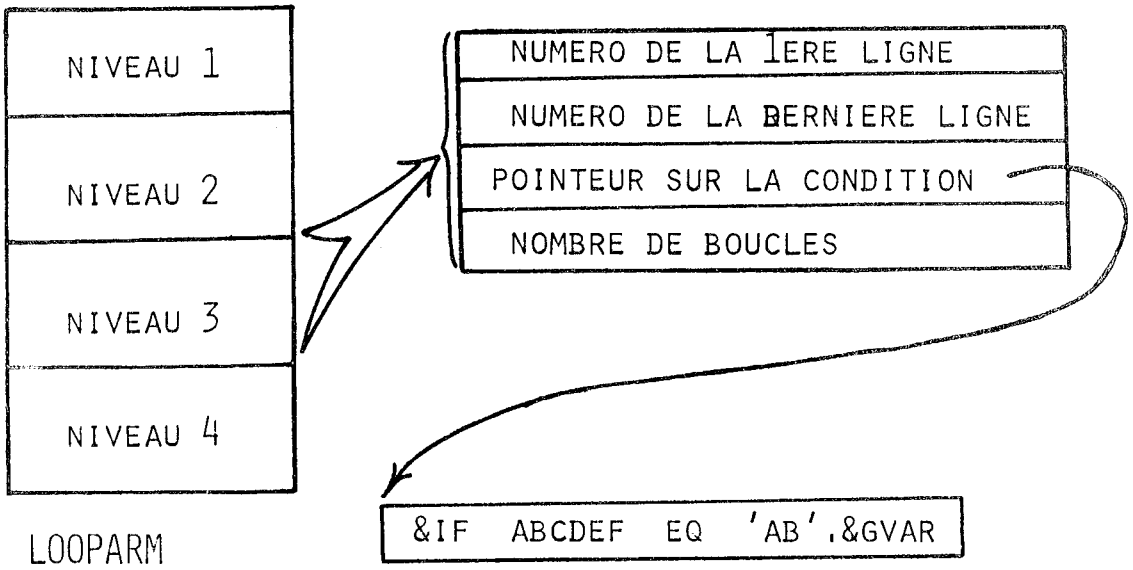
forme non substituée.

Dans le cas de la requête &IF, après avoir déterminé la valeur de la condition, si celle-ci est vraie, on appelle de façon récursive ce module STARTCOM avec en paramètre la ligne correspondant à la partie "instruction" de la requête.

Le traitement de la requête &LOOP commence par une remise à zéro de la variable interne représentant le nombre de lignes restant à lire à la console de l'utilisateur. Le corps de boucle doit en effet exister de façon potentielle afin de pouvoir en déterminer la portée. On détermine alors effectivement cette portée. Le rang de la première ligne de la boucle est égal à celui de la requête &LOOP plus un. Pour déterminer celui de la dernière commande soit on ajoute N-1 au précédent, soit on localise l'étiquette spécifiée selon l'option présente dans la requête. Si la requête est une boucle par condition, on garde cette condition sous forme non substituée. La valeur de celle-ci sera testée avant chaque tour de boucle. Sinon, on garde le nombre de boucles à effectuer.

Tous ces éléments concernant une boucle sont mémorisés dans un tableau à une dimension et comprenant quatre éléments. Chaque élément est une structure qui décrit un niveau de boucle. La représentation interne de ces éléments

est la suivante :



Mémorisation des Paramètres de Boucle

Figure 2

Toutes ces variables sont utilisées par le premier module décrit (EXECTOR) dans la phase où il sélectionne la prochaine commande à interpréter.

3.4.1.2 DESCRIPTION INTERNE DES AFFECTATIONS

Une instruction d'affectation peut se décomposer en trois parties :

- la partie gauche ou "but" de l'affectation.
- la partie droite ou "source" de l'affectation.
- le symbole d'affectation =.

On commence tout d'abord par déterminer la valeur de la

partie droite. Celle-ci peut être d'un des trois types suivants :

- chaîne de caractères : la valeur de la source est la chaîne de caractères commençant au premier caractère non blanc après le signe = et qui se termine au caractère non blanc le plus à droite.
- expression numérique : la valeur de la source est la valeur calculée de l'expression.
- fonction interne : le résultat de l'évaluation de la fonction constitue la valeur de la source.

On détermine ensuite le "but" de l'affectation. On peut avoir un argument, une variable locale, une variable globale.

La représentation interne pour les arguments est un tableau à 31 entrées. Une entrée contient :

- la longueur de la valeur de l'argument.
- un pointeur vers la valeur de l'argument.

Cette zone de valeur est acquise dynamiquement pour chaque argument.

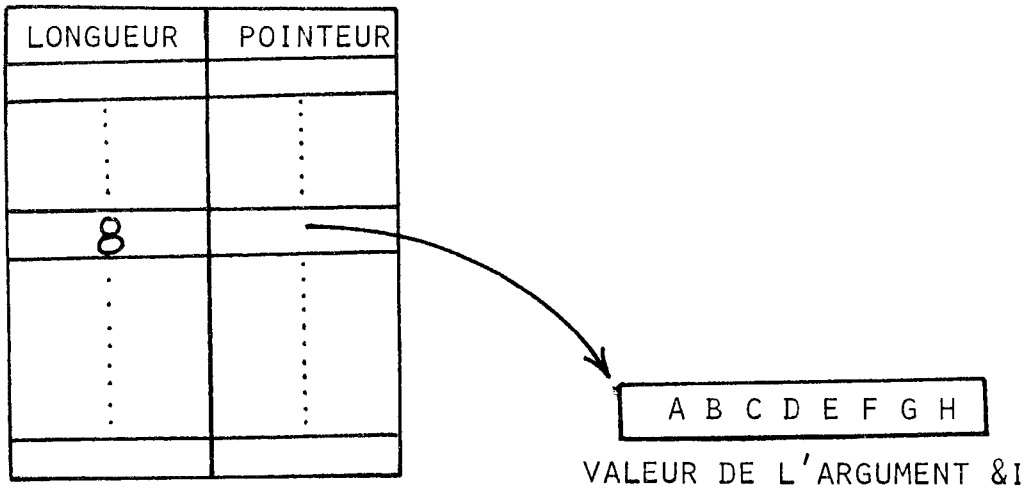


Tableau des Arguments

Figure 3

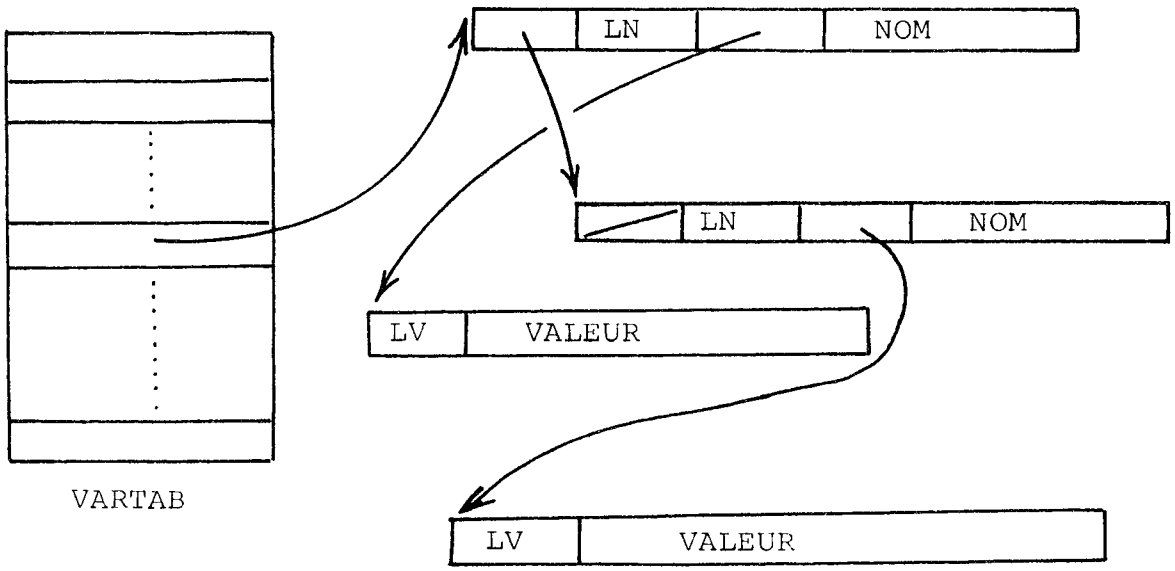
Les variables locales définies par l'utilisateur ont en mémoire une représentation à trois niveaux.

- d'une part une table statique. Chaque élément de cette table sert d'origine pour une liste de variables. On accède à une entrée de cette table par une fonction de hash-code appliquée sur le nom des variables.
- d'autre part, une liste dont chaque élément contient quatre champs :
  - une référence sur l'élément suivant.
  - la longueur du nom de la variable.
  - une référence vers la valeur de la variable.
  - le nom de la variable.
- enfin un élément composé de deux champs :
  - la longueur de la représentation en

caractères de la valeur de la variable.

- la représentation de cette valeur.

L'allocation du premier niveau est statique par rapport au niveau de la macro-commande, mais elle est dynamique vis à vis de l'ensemble des niveaux de récursivité. Les deux autres niveaux sont alloués dynamiquement au niveau de la macro-commande.



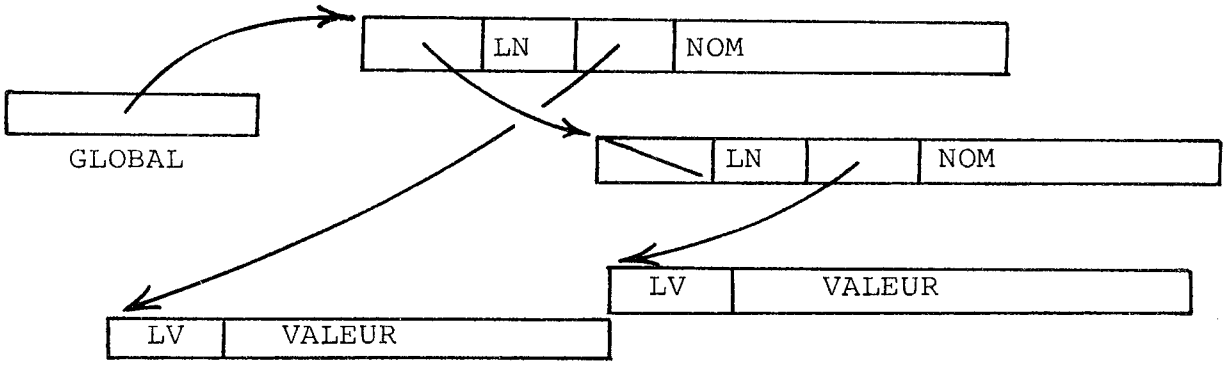
LN : longueur du nom
LV : longueur de la valeur
NOM : représentation du nom
VALEUR : représentation de la valeur

Variables Locales

Figure 4

Enfin les variables globales ont une représentation interne analogue à celle des variables locales, exception faite pour le premier niveau qui n'est constitué que d'une

seule entrée.



Variables Globales

Figure 5

Après avoir déterminé le type du "but", s'il n'est pas encore défini, on alloue une zone pour le nom s'il s'agit d'une variable. Quant à la zone valeur, elle est créée ou réallouée suivant que le "but" n'est pas défini ou que son ancienne valeur a une représentation plus courte que la nouvelle.

Une fois les opérations de définition de la source et d'allocation du "but" réalisées, il y a affectation proprement dite.

3.4.2 INTERPRETATION DES COMMANDES

Nous avons vu que les commandes se distinguaient des requêtes de contrôle par le premier caractère du premier mot, qui n'est pas un &.

Pour lancer l'exécution d'une commande appartenant au langage de commande, nous utilisons le schéma d'appel standard qui est le suivant :

- On donne à la commande la forme d'une liste d'éléments ayant une longueur de huit caractères chacun.
- On marque la fin de liste par un élément de valeur 'FFFFFFFF' en notation hexadécimale.
- On met dans le registre général 1 l'adresse du premier mot de la liste ; c'est à dire du nom de la commande.
- Enfin il y a exécution de la commande SVC 202.

Avant de lancer l'exécution de ces commandes, on examine deux variables d'état. La première renseigne sur les options choisies par l'utilisateur pour la trace de l'interprétation de la macro-commande. La commande sera donc éventuellement imprimée, précédée ou non de l'heure, selon que l'option TIME a été ou non spécifiée avec la requête &CONTROL.

La seconde variable renseigne sur la façon de comptabiliser les temps consommés pour l'exécution de la commande. Si la valeur de cette variable correspond à l'option ON de la requête &TIME, les temps unité centrale (virtuel et total) avant l'exécution de la commande seront retranchés de ceux obtenus après exécution et le résultat sera imprimé. Ce résultat dont la forme est

T=VV.VV/TT.TT HH:MM:SS

exprime le temps dépensé pour l'exécution de la commande ; VV.VV représente le temps virtuel et TT.TT le temps total.

Donnons sur un exemple la façon de lancer une commande. Soit la commande

```
ASSEMBLE MONFICHIERNUM1 (PR NOTERM
```

La liste construite aura la forme :

```
PLIST  ASSEMBLE
       MONFICHI
       (
       PR
       NOTERM
       'FFFFFFFF'
```

On exécute ensuite la séquence d'instructions :

```
LA   1,PLIST
SVC  202
DC   AL4(0)
```

qui est la forme d'appel standard pour toutes les commandes.

Notons que le dernier mot de la séquence d'appel contient l'adresse sur laquelle le programme se déroutera si l'exécution de la commande s'est soldée par une erreur. Si cette adresse est nulle, on continue en séquence.

La trace de l'exécution de cette commande sur la console de l'utilisateur sera :

```
10:32:47 ASSEMBLE MONFICHIERNUM1 (PR NOTERM
NO STATEMENTS FLAGGED IN THIS ASSEMBLY
T=02.57/04.18 10:34:11
```

Le premier message est dû aux options ALL et TIME de la requête &CONTROL. Le second provient du module assembleur. Enfin, le dernier est provoqué par l'option ON de la requête &TIME.

Après l'exécution des commandes, le registre général 15 contient la valeur du code de retour. Celle-ci sera rangée dans la variable spéciale &RETCODE et éventuellement imprimée.

CAS PARTICULIER DE LA COMMANDE EXEC.

Dans le cas particulier où la commande est un appel à une macro-commande, on n'utilise pas le schéma classique d'appel pour les commandes. Il est en effet inutile de repasser par le mécanisme général d'interprétation des commandes puisqu'entre autre, l'interpréteur de macro-commande est déjà chargé en mémoire. Dans ce cas il y

a donc seulement un appel récursif du premier module décrit (EXECTOR), auquel on fournit en paramètre la ligne

EXEC nom-de-macro liste d'arguments

Ceci évite de repasser dans les modules d'initialisation.

C O N C L U S I O N

Dans ce qui précède nous avons présenté les mécanismes et la structure de base de langages de commande, et nous avons proposé quelques réflexions sur un macro-langage.

Cette étude critique des mécanismes, faite plus particulièrement pour le système CMS/370, nous a permis de proposer un macro-mécanisme d'extension du langage de base, enrichi par rapport à ceux existants. C'est la réalisation de ce prototype que nous avons décrite de façon externe dans la seconde partie et de façon interne dans la troisième.

Cependant, comme dans toute réalisation pratique qui est une seconde phase dans laquelle les concepts et les idées avancés dans la première phase sont effectivement implémentés, suit une troisième phase qui consiste à faire une évaluation des résultats obtenus. Celle-ci peut être décomposée en deux parties. D'une part ce que nous appellerons une évaluation fonctionnelle. D'autre part, une partie mesure de performances destinée à mesurer l'efficacité du prototype réalisé.

Celui-ci, utilisé encore de façon expérimentale, semble sur le plan fonctionnel répondre aux objectifs que nous nous étions fixés. La majorité des contraintes d'écriture ont été supprimées à l'intérieur des macro-commandes, ce qui rend le langage facile à utiliser. De plus l'introduction de la notion de chaîne de longueur variable donne, aussi bien au

langage de contrôle qu'au langage de commande, une souplesse et une efficacité meilleure que dans le mécanisme existant. Enfin les facilités offertes par le macro-langage que sont les requêtes de contrôle, les fonctions internes et la manipulation de variables, confère au macro-mécanisme une puissance certaine. Les résultats fonctionnels obtenus semblent donc suffisants pour envisager dans l'avenir une nouvelle conception du système CMS qui serait alors réduit à un certain nombre de primitives que chacun pourrait étendre au gré de ses besoins grâce à ce mécanisme.

Cependant, malgré les qualités fonctionnelles que peut avoir tout mécanisme, pour être viable, il doit être aussi peu coûteux que possible en termes de ressources consommées. Et si maintenant les contraintes de taille semblent diminuer du fait de l'apparition de technologies nouvelles et de concepts nouveaux qui rendent les mémoires plus vastes et moins coûteuses (mémoires virtuelles), c'est surtout l'aspect temps consommé qui importe. Et sur ce plan, nous ne pouvons que constater une dégradation des performances par rapport au mécanisme déjà existant dans CMS.

Une des premières raisons provient du langage utilisé dans la réalisation de notre prototype. Comme nous l'avons mentionné au chapitre 3, le langage utilisé pour l'écriture a été un langage de haut niveau, comparable à PL/I, qui génère du code assembleur. La quantité de code généré est de

toute évidence supérieure à celle que nous aurions obtenu en écrivant directement notre mécanisme en langage assembleur 370. Ce qui ne peut qu'induire une charge plus forte.

La seconde cause, la plus importante, provient du degré de généralité offert qui a été introduit par la notion de chaînes de caractères de longueur variable. Si les avantages fonctionnels offerts sont incontestables, l'efficacité s'en trouve fortement pénalisée. Notons d'ailleurs que P.J. BROWN mentionnait déjà ce problème en disant "que la charge introduite par le traitement des chaînes de caractères pouvait rendre un macro-processeur trop lent pour être un outil pratique".

Tout travail de recherche ne constitue pas une fin en soi. Ce travail de minimisation du coût constitue donc une ouverture vers l'avenir. Nous devrons pour cela commencer par faire des mesures très précises du mécanisme afin de localiser les points coûteux, et plus particulièrement faire une évaluation du coût introduit par le langage de programmation et les chaînes de caractères. Il faudra ensuite appliquer les remèdes nécessaires. Ceci pourra être, outre un réécriture dans un langage de bas niveau et le resserrement de certaines généralités, la production d'un code intermédiaire qui permettrait de tenir compte des éléments invariants dans une commande. Enfin, après une conception précise et rigoureuse, nous pourrons envisager

une assistance micro-programmée pour le mécanisme. C'est ce à quoi nous essaierons de nous attacher.

B I B L I O G R A P H I E

ASSABGUI — M. ASSABGUI, Assembleur et Macro-Assembleur 360, Cours Polycopié, Université Scientifique et Médicale de Grenoble.

AUROUX-HANS — A. AUROUX et C. HANS, Editeur de fichiers et macro-langage d'un système conversationnel, Colloque international sur l'informatique, Mars 1969

BELLOT — M. BELLOT et al, Systèmes de Programmation Générateurs de Machines Virtuelles, Cours Polycopié, Grenoble 1973

BROWN — P.J. BROWN, A Survey of Macro-Processors, Automatic Programming, VOL.6

CREVEUIL-73 — M. CREVEUIL, EXEC un Macro-Processeur pour CMS, Rapport de DEA, Université Scientifique et Médicale de Grenoble, juin 1973.

CREVEUIL-74 — M. CREVEUIL, Contrôle des Commandes dans un Système Conversationnel, Prix Concours Etudiant Chapitre Français de l'ACM, avril 1974

DECALUWE — R. DECALUWE, Etude et Réalisation d'un langage de Commande pour un Réseau d'Ordinateurs, Thèse de 3e cycle, Université de Grenoble 1973.

DOHERTY — W.J. DOHERTY, C.H.THOMPSON, S.J. BOIES, An Analysis of Interactive System Usage with Respect to Software, Linguistic and Scheduling

Attributes

DOLOTTA — T.A. DOLOTTA & C.A. IRVINE, Proposal for a Time-Sharing Command Structure, IFIP Congress, 1968, C14.

GRANT — C.A. GRANT, An Interactive Command Facility, JACM, juillet 1970.

GURSKI — A.F. GURSKI, Towards a High-Level Job Control Language, Colloque sur la Programmation, Institut de Programmation de Paris, avril 1974

HANS — C. HANS, Contribution à l'Architecture de Mécanismes Elémentaires pour certains Systèmes Générateurs de Machines Virtuelles, Thèse de Docteur ès-Sciences, Université de Grenoble Novembre 1973.

IBM CMS — IBM CMS User's Guide GH20-0859

IBM CP — IBM Virtual Machine Facility/370: Control Program (CP) Program Logic SY20-0880-1

IBM EXEC — IBM Virtual Machine Facility/370: EXEC User's Guide, GC20-1812-0

IBM OS — OS/VS1 JCL Reference GC24-5099-0

IBM VM — IBM Virtual Machine Facility/370: Command Language User's Guide, GC20-1804-0

LEAVENWORTH — B.M. LEAVENWORTH, Syntax Macros and Extended Translation, ACM, novembre 1966

LE HEIGET — J.P. LE HEIGET, Généralisation de la Notion d'espaces Virtuels sous les Systèmes CP-67/CMS, Thèse CNAM, Université de Grenoble, juillet 1972

LEROUDIER — J. LEROUDIER, Une Analyse de Système, Thèse de 3e cycle, Université de Grenoble, avril 1973

SAVARY — H. SAVARY et M. BERTAUD, Etude et Réalisation d'un Macro-Langage Conversationnel MALIN, Rapport de DEA, Université de Grenoble 1971

SNOBOL — R.E. GRISWOLD, J.F. POAGE, I.P. POLONSKY, The SNOBOL 4 Programming Language, Prentice Hall 1974 (2e édition)

STEPHENSON — C.J. STEPHENSON, On the Structure and Control of Commands, 4ième Symposium sur les Operating Systems, Stanford, Californie, Octobre 1973

VIDART-73 — J. VIDART, Macros Syntaxiques et Langages de Programmation Système, Etude no FF2-0150, Centre Scientifique IBM de Grenoble, janvier 1973

VIDART-74 — J. VIDART, Extension Syntaxique et Langage LL(1), Thèse 3e cycle (à paraître), Université de Grenoble.