XCP : un environnement graphique conversationnel pour l'examen des structures d'un système , application à CP-67

Thanh Thi Nguyen

N° d'ordre

THESE

présentée à

L'Université Scientifique et Médicale de Grenoble

pour obtenir

LE GRADE DE DOCTEUR DE TROISIEME CYCLE

« Informatique »

par

NGUYEN THANH THI

°°°°°

XCP UN ENVIRONNEMENT GRAPHIQUE CONVERSATIONNEL POUR L'EXAMEN DES STRUCTURES D'UN SYSTEME APPLICATION A CP-67

°°°°°

Thèse soutenue le 6 Juin 1973 devant la commission d'examen

Monsieur	N. GASTINEL	Président
Messieurs	L. BOLLIET	Examinateurs
	M. GRIFFITHS	
	C. HANS	

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
	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
	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
	BRAVARD Yves	Géographie
	BRISSONNEAU Pierre	Physique du Solide
	BUYLE-BODIN Maurice	Electronique
	CABANAC Jean	Pathologie chirurgicale
	CABANEL Guy	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
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée
	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

MM.	GASTINEL Noël	Analyse numérique
	GERBER Robert	Mathématiques Pures
	GIRAUD Pierre	Géologie
	KLEIN Joseph	Mathématiques Pures
Mme	KOFLER Lucie	Botanique et Physiologie végétale
MM.	KOSZUL Jean-Louis	Mathématiques Pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques Appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie pharmaceutique
	LAURENT Pierre	Mathématiques Appliquées
	LEDRU Jean	Clinique médicale B
	LLIBOUTRY Louis	Géophysique
	LOUP Jean	Géographie
Mlle	LUTZ Elisabeth	Mathématiques Pures
	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	Servomécanismes
	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
	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLAND François	Zoologie
	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
	RADHAKRISHNA Pidatala	Thermodynamique

PROFESSEURS SANS CHAIRE

MM. AUBERT Guy	Physique
BEAUDOING André	Pédiatrie
BERTRANDIAS Jean-Paul	Mathématiques Appliquées
BIARES 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
DESRE Pierre	Métallurgie
DOLIQUE Jean-Michel	Physique des Plasmas
GAUTHIER Yves	Sciences biologiques
GEINDRE Michel	Electroradiologie
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
SARRAZIN Roger	Anatomie et chirurgie
SARROT-REYNAULD Jean	Géologie
SIBILLE Robert	Construction Mécanique
SIROT Louis	Chirurgie générale
Mme SOUTIF Jeanne	Physique générale
M. VALENTIN Jacques	Physique nucléaire

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
Mme BERTRANDIAS Françoise	Mathématiques Pures
MM. BLIMAN Samuel	Electronique (EIE)
BLOCH Daniel	Electrotechnique
Mme BOUCHE Liane	Mathématiques (CUS)
MM. BOUCHET Yves	Anatomie
BOUSSARD Jean-Claude	Mathématiques Appliquées
BOUVARD Maurice	Mécanique des Fluides
BRIERE Georges	Physique expérimentale
BRODEAU François	Mathématiques (IUT B)
BRUGEL Lucien	Energétique
BUISSON Roger	Physique
BUTEL Jean	Orthopédie
CHAMBAZ Edmond	Biochimie médicale
CHAMPETIER Jean	Anatomie et organogénèse

MM. CHIAVERINA Jean	Biologie appliquée (EFP)
CHIBON Pierre	Biologie animale
COHEN-ADDAD Jean-Pierre	Spectrométrie physique
COLOMB Maurice	Biochimie médicale
CONTE René	Physique
CROUZET Guy	Radiologie
DURAND Francis	Métallurgie
DUSSAUD René	Mathématiques (CUS)
Mme ETERRADOSSI Jacqueline	Physiologie
MM. FAURE Jacques	Médecine légale
GAVEND Michel	Pharmacologie
GENSAC Pierre	Botanique
GERMAIN Jean-Pierre	Mécanique
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
Mme LAJZEROWICZ Jeannine	Physique
MM. LAJZEROWICZ Joseph	Physique
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)
MAYNARD Roger	Physique du Solide
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 PIERY Yvette	Biologie animale
MM. RACHAIL Michel	Médecine interne
RACINET Claude	Gynécologie et obstétrique
RICHARD Lucien	Botanique
Mme RINAUDO Marguerite	Chimie macromoléculaire
MM. ROMIER Guy	Mathématiques (IUT B)
ROUGEMONT (DE) Jacques	Neuro-Chirurgie
STIEGLITZ Paul	Anesthésiologie

MM. 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
YACOUD Mahmoud	Médecine légale

CHARGES DE FONCTIONS DE MATIRES DE CONFERENCES

Mme BERIEL Hélène	Physiologie
Mme RENAUDET Jacqueline	Microbiologie

Fait le 8 MARS 1972.

J'exprime mes sincères remerciements à :

Monsieur le Professeur Noël GASTINEL, Directeur du Centre Interuniversitaire de Calcul de Grenoble, qui a bien voulu me faire l'honneur de présider le Jury de cette thèse.

Monsieur Louis BOLLIET, Professeur à l'Institut Universitaire de Technologie de Grenoble, qui a dirigé cette thèse et m'a toujours témoigné sa confiance et sa sympathie.

Monsieur Michaël GRIFFITHS, Maître de Conférence à l'Université Scientifique et Médicale de Grenoble, pour l'intérêt qu'il a porté à mon travail et qui a aimablement accepté de faire partie du Jury.

Monsieur Claude HANS, Ingénieur au Centre Scientifique IBM de Grenoble, qui est à l'origine de cette thèse et a su me conseiller et m'encourager tout au long de la réalisation de mon travail.

Messieurs Maurice BELLOT et Jean-Pierre LEHEIGET pour leurs précieuses suggestions et pour les nombreuses critiques et corrections apportées au manuscrit.

Tous mes collègues et amis du Laboratoire qui m'ont aidé dans mon travail, en particulier Madame Liliane SIRET et Messieurs Jean-Pierre DUPUY, Jacques GRANIER, Jean GUILLOU, Pierre HERNICOT, Michel LUCAS et Maurice REY.

Mademoiselle Corinne SALLUSTIO et le Service de Reproduction pour la réalisation soignée et rapide de ce document.

S O M M A I R E

C H A P I T R E 1

INTRODUCTION

D'année en année les systèmes de contrôle des ordinateurs de grande taille sont de plus en plus élaborés. Du point de vue de l'utilisateur, cette évolution se traduit surtout par les possibilités accrues des installations informatiques auxquelles il a accès. Par contre, le travail du personnel responsable de la maintenance d'un système a perdu en simplicité; il est dans la plupart des cas, rendu encore plus délicat par le manque d'outils appropriés.

Notre travail porte principalement sur ce dernier aspect du problème de maintenance; mais par certains côtés, il offre également des possibilités pour l'enseignement d'un système et en particulier de ses structures internes.

1 - MAINTENANCE D'UN SYSTEME -

La maintenance d'un système d'exploitation est dûe à la nécessité de son évolution afin d'améliorer ses performances, ou à l'adjonction de nouveaux composants. Ceux-ci sont destinés, soit à faciliter le travail des utilisateurs soit à gérer des nouveaux éléments "hardware" ou "software".

Ce travail de maintenance n'est guère facile en raison des relations complexes entre les différents modules du système et du volume, sans cesse croissant, des instructions qui le composent. Dans le cas du système TSS/360 par exemple le nombre de modules est passé de 700 (version 1.0) à 1700 (version 8.1). Les responsables de la maintenance doivent suivre de près cette évolution sous peine de ne plus pouvoir assurer parfaitement leur travail, car ils peuvent introduire des incompatibilités entre

les modules.

Dans le problème de maintenance nous distinguons également celui de la correction des erreurs inhérentes à chaque version. La sortie d'une version d'un système est précédée d'une période de test afin d'éliminer les erreurs "software". Mais la correction de ces erreurs peut entrainer l'introduction de nouvelles erreurs dont les effets sont parfois à retardement. Ainsi on a estimé que chaque nouvelle version d'OS/360 contient environ un millier d'erreurs et que ce nombre augmente avec les versions ultérieures. Une statistique précise a été faite par R.SCHEWMM [Ref.35] pour le système TSS/360. Le tableau ci-dessous donne les résultats de cette étude :

Version	Nombre de modifications apportées	Nbre d'erreurs détectées et corrigées par :	
		Les utilisateurs	IBM
1.1	9	91	-
1.2	32	136	7
2.0	49	181	72
3.0	56	190	255
4.0	42	111	201
5.0	80	321	534
5.1	32	192	471
6.0	80	226	538
6.1	-	40	85
7.0	64	174	466
8.0	50	206	442
8.1	70	574	1016
TOTAL	568	2442	4087

Ces chiffres posent bien le problème de la détection et de la correction des erreurs dans un système d'exploitation. Il est en effet pratiquement impossible d'attendre que toutes les erreurs soient supprimées pour livrer la version en exploitation.

2 - ENSEIGNEMENT D'UN SYSTEME -

Le problème de la formation des nouveaux spécialistes en système est aussi important que le problème de la maintenance. Il est à peu près impossible à une seule personne d'être véritablement très bien informée de la logique interne d'un système dans son intégralité. Cette difficulté provient toujours du fait de la complexité et de la taille des systèmes.

Aucune solution véritable de ces problèmes ne sera satisfaisante tant que l'on ne saura pas minimiser cette complexité pour mieux la maitriser.

Du point de vue des réalisations pratiques, divers simulateurs de machines ont été développés et ont beaucoup aidé à la réalisation et au développement des systèmes. Certaines erreurs ont été ainsi éliminées grâce à ce genre d'outil, mais d'autres, en particulier celles dépendantes du temps, n'ont pû être localisées et corrigées. Quant aux outils "software" spécifiquement système, un seul à notre connaissance a été développé. C'est TSSS (Time Sharing Support System) [Ref.23], sous-système de TSS/360, permettant à un programmeur système de détecter, d'analyser les erreurs et de modifier dynamiquement les modules en cause, ceci grâce à un ensemble de requêtes intéractives et une description symbolique des objets manipulés. Toutefois, il faut remarquer que la mise en oeuvre d'un tel "software" présente de nombreux inconvénients :

- l'activation de TSSS arrête complètement l'activité du système TSS/360.

- le langage des requêtes et de description est assez hermétique :

SET $IO(X'190',4,4,15).(32,8)

- les structures internes du système (blocs de contrôle, listes, etc...) sont difficilement accessibles en raison de la rigidité du langage de description.

Ces problèmes généraux de maintenance apparaissent dans tous les systèmes actuels et sont résolus de façon plus ou moins efficace. Pour notre part nous avons voulu créer une aide intéractive d'emploi aisé pour la détection des erreurs dans le système CP-67. Ce choix est motivé par :

- la notion naturelle de machine virtuelle dans CP-67, facilitant ainsi l'implantation de nouveaux outils dans le système. (Dans les systèmes MTS et TSS/360, cette notion a été introduite seulement plusieurs années après leur sortie).

- la simplicité relative de CP-67 par rapport aux autres systèmes, dûe à la nette séparation de la partie CP-67 (programme de contrôle du 360/67 et de génération des machines virtuelles) et de la partie système d'exploitation fonctionnant sur les différentes machines virtuelles.

- la facilité d'accès à ce système puisqu'il est actuellement actif neuf heures par jour au CICG.

C H A P I T R E 2

SPECIFICATIONS DE L'ENVIRONNEMENT XCP

C H A P I T R E 2

CP-67 est un système générateur de machines virtuelles [Ref 10], qui simule pour chaque utilisateur le comportement d'un ordinateur IBM 360. Le rôle de CP-67 est de gérer les ressources physiques d'un 360/67 (disques, mémoire, etc...) et de les partager entre les différents utilisateurs. Comme tout système superviseur pour ordinateur de grande puissance, CP-67 comporte un grand nombre de modules ayant des relations complexes.

2.1 POSITION DU PROBLEME

Ces relations amènent des difficultés pour la maintenance du système par exemple : correction des erreurs, amélioration du système, adjonction de nouveaux modules. Toutes ces opérations conduisent à réaliser une série de tests avant la mise en service de la nouvelle version du système. Mais, malgrè les précautions prises, les erreurs peuvent toujours se produire et provoquer une dégradation des performances, un blocage, ou encore un arrêt complet du système. L'expérience montre qu'en général ces erreurs sont répétitives lors d'une session de CP-67 et obligent dans les heures qui suivent, la remise en service de l'ancienne version. Du point de vue de l'exploitation, ces arrêts sont assez graves et ressentis par les utilisateurs du fait du temps partagé, contrairement à un système de type batch où les pannes "système" sont presque ignorées par les utilisateurs, puisque ces derniers n'ont aucune intéraction avec leur programme.

2.1.1 Les pannes "système" dans CP-67

Une panne "système" est une erreur détectée, soit par hardware, soit par software, durant l'activité du système. Elles peuvent être bénignes et entrainent alors une dégradation des services rendus

aux utilisateurs ou encore fatales et provoquent dans ce cas un arrêt brutal du système.

2.1.1.1 Les pannes dûes au hardware

Les erreurs hardware proviennent du fonctionnement défectueux des différents organes composant l'ordinateur, par exemple des erreurs d'entrée/sortie, pannes de l'unité centrale etc...

a) Erreurs d'entrée/sortie

Dans CP-67 comme dans tout système les erreurs d'entrée/sortie ne provoquent pas d'arrêt du système mais peuvent entrainer une dégradation des performances, du fait du processus de correction des erreurs. En effet, lorsqu'une erreur d'entrée/sortie est détectée, un programme de rattrapage d'erreur est mis en fonction pour relancer l'entrée/sortie un certain nombre de fois. Ceci peut provoquer une file d'attente sur l'unité d'entrée/sortie, si l'erreur persiste.

Plusieurs zones sur unité à accès direct sont prédéfinies pour recueillir les informations relatives aux erreurs dûes au hardware. Les données enregistrées sont ensuite exploitées pour la maintenance de la machine, l'établissement de statistiques, etc...

b) Pannes de l'unité centrale

Les pannes de l'unité centrale ne peuvent que conduire à arrêter le déroulement de CP-67.

2.1.1.2 Les pannes dûes au software

Elles peuvent être rangées dans 2 catégories :

a) Erreurs de programmation

Ces erreurs sont détectées par le hardware et sont dûes au non-respect des règles de programmation de la machine, par exemple exécution d'une instruction de code inexistant, manipulation d'une donnée non conforme à la définition, etc... Elles se traduisent par une interruption de type programme [Ref 11] qui entraine l'arrêt du système.

b) Erreurs d'algorithme

Nous réunissons sous ce vocable les erreurs dûes au non-respect des conventions du système et les erreurs résultant d'un défaut de conception (cas particulier non prévu, conditions spéciales dépendantes du temps etc....). Elles entrainent des perturbations dans le système, parfois longtemps avant leur détection.

La présence des conditions d'erreur est détectée par des séquences spéciales d'instructions placées aux points critiques du système. Ces séquences vérifient surtout la cohérence des informations manipulées et déclenchent un processus d'arrêt immédiat si certaines de ces informations sont douteuses.

Il est évident que la cause profonde de ces erreurs est longue à trouver, car il faut reconstituer les actions entreprises par le système avant l'issue fatale.

2.1.2 Processus d'obtention d'une image mémoire

Suite à une erreur "système" quelconque, ou à une dégradation des performances, l'état actuel de la machine doit être enregistré pour tenter de déterminer les origines du fonctionnement anormal : c'est le

processus d'obtention d'une image mémoire. Il peut être soit automatique soit manuel, selon le mode de détection des anomalies de fonctionnement.

Processus automatique

Une erreur "software" peut être détectée par le "hardware" de la machine (erreur de type programme) ou par une séquence d'instructions qui déclenche un appel au superviseur (SVC 0) [Ref 14]. A la suite de cette détection un module spécial du système prend une image mémoire de la machine.

Processus manuel

Une image mémoire peut être obtenue sans qu'il y ait une panne du système; en effet, lorsque les performances semblent se dégrader, il est toujours possible de donner manuellement le contrôle au module qui récupère cette image.

En fonction des paramètres de génération du système ou d'exploitation immédiate, cette image peut être copiée sur un disque, sur une bande magnétique ou sur une imprimante.

a) *Dump sur disque*

L'image de la mémoire au moment de la panne est copiée sur disque dans un fichier de "spooling" [Ref 14] ; elle est destinée à une machine virtuelle dont le nom est spécifié au moment de la génération du système.

b) *Dump sur bande ou imprimante*

Les informations contenues dans la mémoire centrale sont écrites sous forme hexadécimale et en caractères EBCDIC équivalents. Chaque enregistrement ou ligne représente le contenu de 32 octets de mémoire. C'est le "dump classique" bien connu des utilisateurs des systèmes de type batch.

REMARQUE

> Dans la suite du texte, nous appelons image mémoire (dump) l'image de la mémoire centrale après une panne du système et instantané (snapshot) l'image de la mémoire prise pendant le fonctionnement normal du système. Leur forme est la même, la seule différence se situe au niveau de leur obtention.

2.1.3 Les outils actuels pour la mise au point et la recherche des erreur

Pour pallier aux pannes "software" du système CP-67 (nous nous intéressons seulement à celles-ci), divers outils sont disponibles:

. avant la mise en service du système pour trouver et corriger le maximum d'erreurs possibles (mise au point).

. après une panne du système, pour aider à chercher les causes de l'arrêt.

a) Outil de mise au point

CP-67 de par sa conception contient son propre outil de mise au point: c'est le fait de pouvoir générer des machines virtuelles et spécialement le modèle 67 avec son dispositif de translation dynamique des adresses [Ref 12].

Cette possibilité du système facilite grandement le développement des futures versions, en faisant fonctionner sous CP-67, une machine virtuelle modèle 360/67 avec comme système, le nouveau système CP-67 à tester. Ceci évite l'immobilisation de la machine réelle pour les tests "système" et favorise la mise au point des nouvelles versions. Il est à noter que cette possibilité permet de supprimer la plus grande partie des erreurs, mais certaines autres (évidemment les plus sournoises) n'apparaissent que lors de tests effectués en "réel".

Ce mode de test dit "en virtuel" offre les possibilités de trace et de point d'arrêt. En effet, l'utilisateur d'une machine virtuelle peut spécifier à CP-67 :

- les adresses des différents points d'arrêts qu'il désire et suspendre ainsi le déroulement du système en test sur la machine virtuelle

- les différents types de traces attendues, à savoir : historiques des instructions de rupture des séquences, des appels au superviseur (instruction SVC) des instructions d'entrée/sortie, et des interruptions externes , programme et entrée/sortie.

b) Aide à la recherche des erreurs

A la suite d'une défaillance de CP-67, le programmeur système dispose pour son travail de recherche et d'analyse des erreurs, d'une image mémoire sous la forme d'une liasse de papier (dump classique de la mémoire traduit en caractères EBCDIC) ou du contenu d'un fichier de "spooling" (en binaire non traduit). L'utilisation de cette dernière

forme permet d'accélérer le processus de "dump" à l'instant de la défaillance du système, mais ce contenu devra plus tard être traduit.

Pour exploiter l'image mémoire contenue dans un fichier de "spooling", deux programmes du système CP-67 sont actuellement disponibles :

FDUMP et EDITDUMP permettant l'un de lire le contenu du fichier de "spooling" et l'autre de l'écrire, après traduction, sur une imprimante. Quant à la recherche de la cause de l'erreur, elle consiste en un examen visuel du contenu de ces feuilles de papier. Les seules aides dont le programmeur système peut disposer sont : les listes d'assemblage des modules de CP-67, sa connaissance du système et bien sûr son intuition.

2.2 LES OBJECTIFS VISES

Devant la complexité des systèmes de programmation, un programmeur système se trouve submergé par la quantité d'information qu'il a à dépouiller pour chercher les causes d'une erreur. De même, une personne voulant étudier un système doit acquérir une somme de connaissances très importante par exemple par la lecture des brochures, et ceci sans qu'il puisse avoir une idée exacte et concrète du système.

Pour remédier à cet état de fait nous avons conçu et réalisé un environnement spécialisé fonctionnant de façon intéractive et qui tente de répondre aux objectifs suivants :

- aider le plus possible le programmeur système dans son travail de recherche des causes d'erreur software ou de dégradation des performances du système CP-67.

- permettre au programmeur non spécialiste de connaître et d'apprendre la logique interne de CP-67.

Cet environnement appelé XCP est constitué de plusieurs sous-environnements ayant chacun une fonction déterminée et indépendante des autres.

2.2.1 Aide à la recherche des erreurs

Pour le système CP-67, nous avons vu qu'il n'existe pratiquement pas d'aide programmé lorsque se produit une défaillance du système. Grâce à XCP, le programmeur système peut prendre une photographie de CP-67 (image mémoire ou instantané) afin de l'examiner de façon symbolique et intéractive. En effet, par l'intermédiaire d'un langage de description des données, il peut décrire toutes les variables de CP-67 (nom, type, relation et, grâce à un langage de requêtes, donner des ordres à XCP pour l'aider dans son travail d'examen. XCP n'est pas un outil capable de trouver seul des erreurs, mais il donne des moyens très précieux dans le travail d'investigation et de diagnostic. La détermination des causes d'une erreur sera toujours liée à l'habileté du programmeur système; toutefois XCP facilite son travail et diminue le temps de recherche en réalisant pour lui les opérations machinales et fastidieuses.

2.2.2 Aide à la connaissance d'un système

Ayant quelque peu l'expérience de l'enseignement de la logique d'un système de programmation nous avons constaté que cet enseignement se compose de 2 phases :

- d'une part l'explication de divers algorithmes utilisés.

- d'autre part la présentation des structures de données manipulées par ces algorithmes.

Ce dernier aspect est généralement rébarbatif à enseigner, car il consiste à apprendre le format des nombreuses tables et blocs, décrits dans un langage non évolué, et à examiner des dizaines de pages de dump du système étudié.

Avec l'environnement XCP proposé, nous pensons par exemple, pouvoir montrer à un étudiant les structures des données utilisées, l'accès à ces données, les contenus, etc...Ceci par une description assez proche de celles utilisées dans les langages de haut niveau qu'il connait et qui donc, ne le rebutera pas. Grâce à la description symbolique des variables, l'environnement XCP pourra être classé comme un système de type "question-réponse" avec lequel l'intéressé a la possibilité d'obtenir des informations sur le système étudié.

2.3 PRESENTATION DE L'ENVIRONNEMENT XCP

L'environnement XCP fonctionne sur une machine virtuelle et ne dépend de CP-67 que pour les données à traiter. Il peut être facilement adapté à un autre système de programmation.

La figure (2.1) montre les différents dispositifs d'entrée/sortie utilisés, en particulier le terminal graphique IBM 2250 [Réf 13] dont la rapidité facilite l'intéraction entre l'utilisateur et l'environnement XCP. Le dialogue se fait à partir d'un terminal classique (machine à écrire) ou du terminal graphique, unités pour lesquelles l'utilisateur

exprime ses ordres à XCP en frappant un message (requête) ou en actionnant une touche du 2250 (fonction programmée) ou encore en utilisant le pointeur optique (requête graphique).

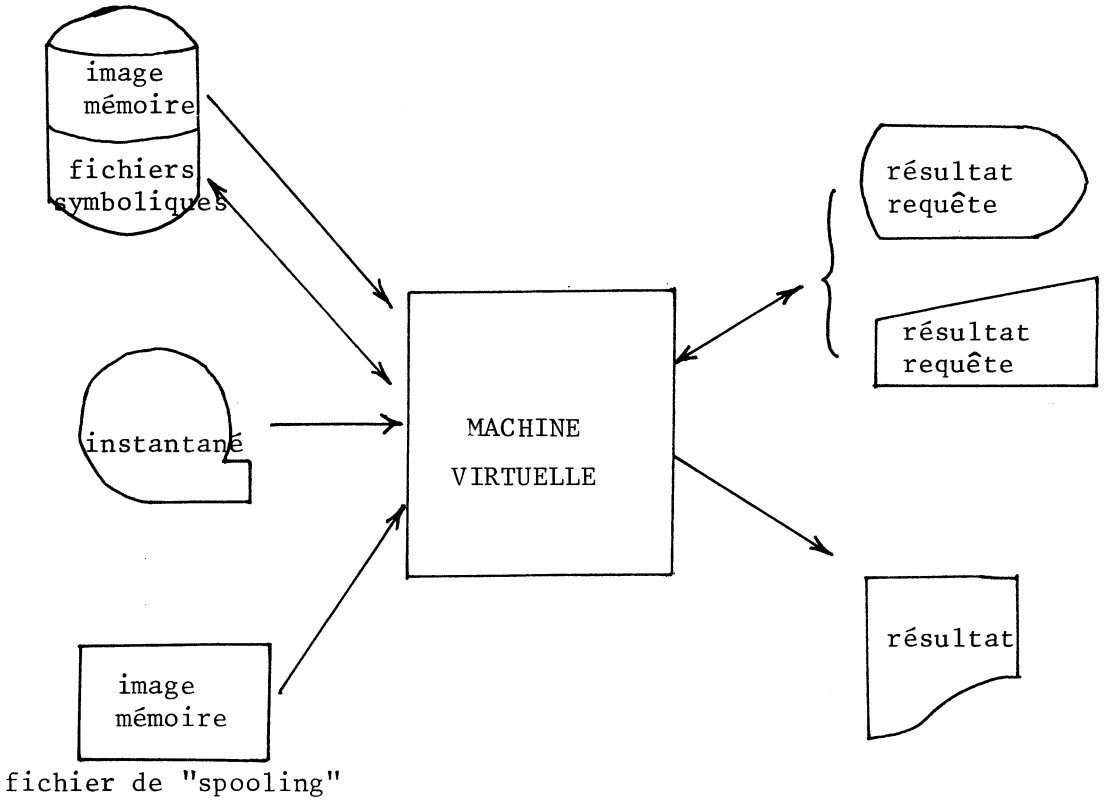


Figure 2.1

Dispositifs d'entrée/sortie

2.3.1 Les différents sous-environnements de XCP

Au point de vue logique, XCP se compose de 4 sous-environnements ayant chacun un rôle déterminé (fig.2.2). Le passage d'un sous environnement à un autre se fait soit par une requête, soit par une fonction demandée à l'aide d'une touche du 2250.

L'environnement peut être comparé à un automate d'état finis dont les différents états sont les sous-environnements.

2.3.1.1 Le sous-environnement de définition

Dans ce sous-environnement de XCP, l'utilisateur dispose d'un ensemble de requêtes lui permettant de définir, modifier, etc... les structures de données représentant la structure interne du système CP-67. Toute variable référencée ultérieurement doit être déclarée dans ce sous-environnement.

Nous avons créé un langage de description des données beaucoup mieuxadapté que le langage d'assemblage du 360, pour décrire les variables de CP-67. Chaque message reçu par le sous-environnement de définition fait l'objet d'une vérification syntaxique immédiate, tandis que la vérification sémantique n'est faite qu'au moment du passage de ce sous-environnement dans celui d'analyse. En effet,cette dernière vérification ne peut se faire que si toutes les variables sont définies, en raison des relations possibles entre elles.

2.3.1.2 Le sous-environnement de modification

Ce sous-environnement n'est accessible qu'à partir du précédent et permet de corriger les déclarations des variables structurées (§ Chapitre 3) précédemment définies. Il se comporte comme un éditeur

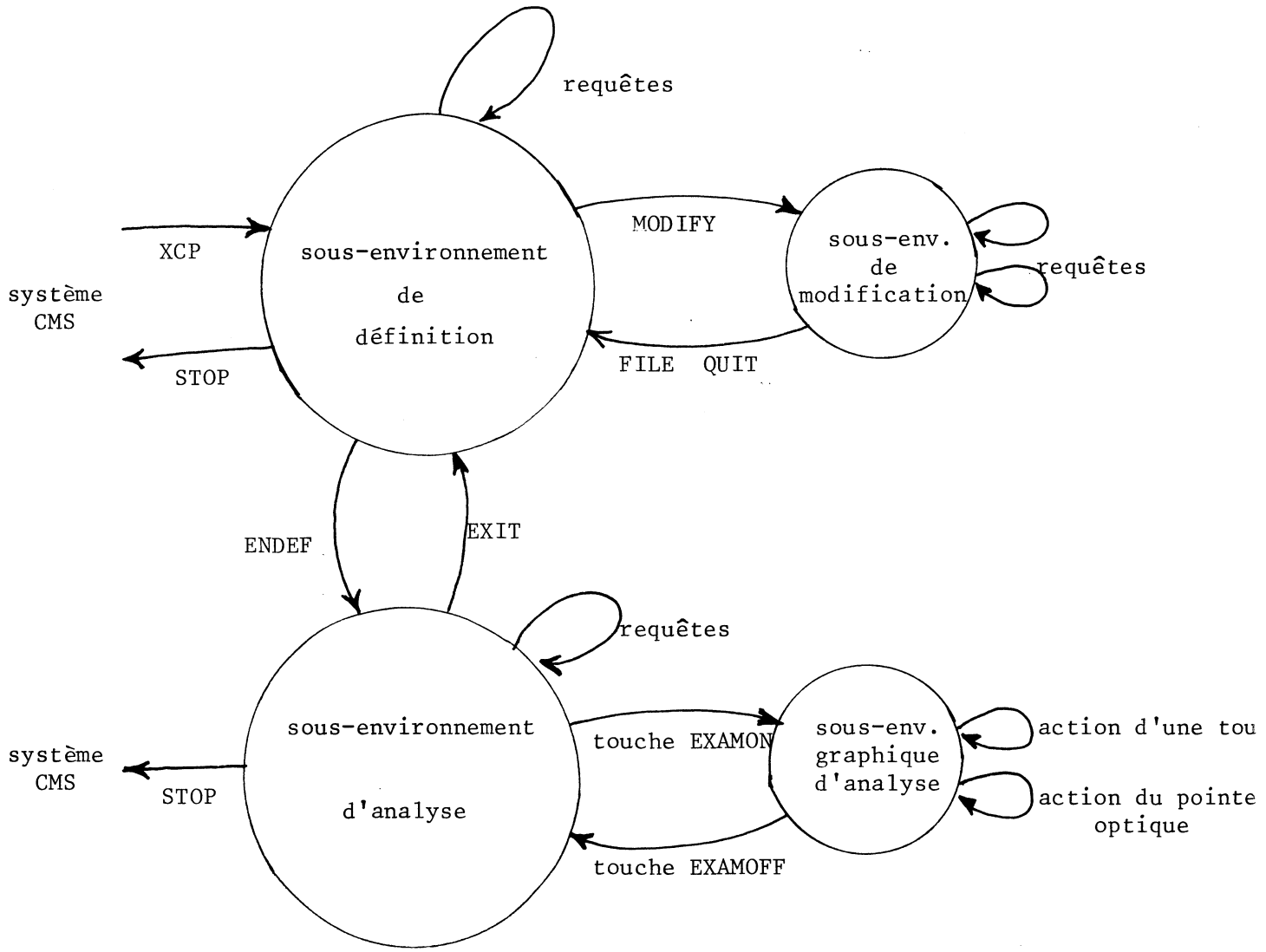


Figure 2.2

Les sous-environnements

de texte spécialisé, car chaque modification apportée, nécessite une vérification de la syntaxe du nouvel élément de la structure.

2.3.1.3 Le sous-environnement d'analyse

Ce sous-environnement n'est accessible que si l'ensemble des variables que l'on va utiliser est défini et qu'aucune erreur n'est détectée lors de la vérification sémantique.

Le sous-environnement d'analyse comporte essentiellement des requêtes permettant un examen de l'image mémoire ou de l'instantané. Cet examen se fait en amenant les données en question (image mémoire de CP-67) dans la mémoire même de la machine virtuelle, en appliquant un facteur de translation T. (fig 2.3).

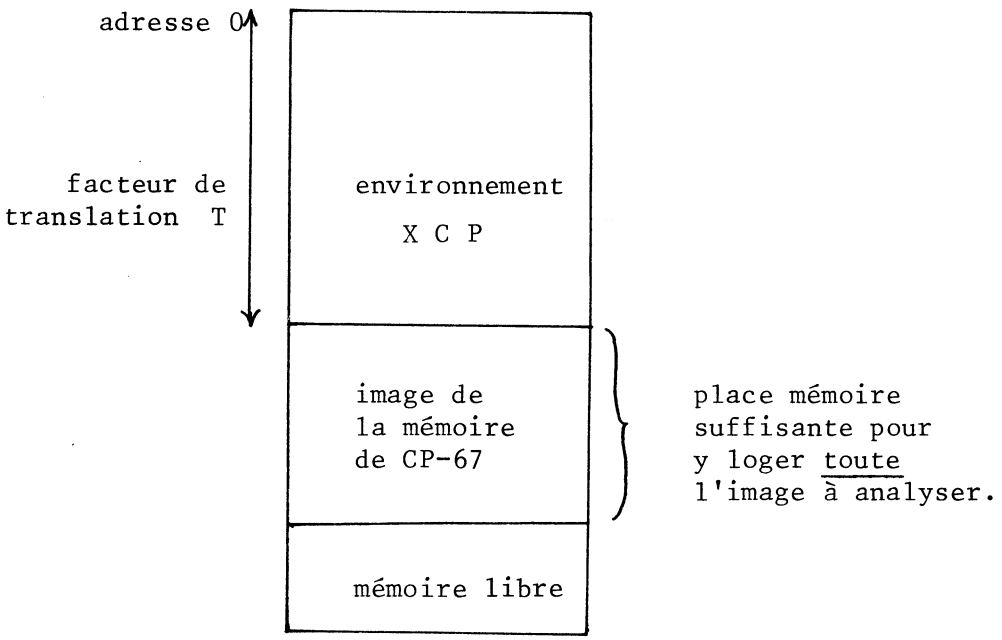


Figure 2.3

Utilisation de la mémoire virtuelle

Ainsi l'emplacement de toute référence à la mémoire de CP-67 est facilement calculée en ajoutant à cette référence le facteur de translation. Cette solution est préférée à celle qui consiste à amener , page après page l'image mémoire de CP-67 à partir du support à accès direct sur lequel elle réside; dans ce cas, l'analyse conduirait à :

- interpréter toutes les références pour savoir si leur emplacement est en mémoire

- calculer dans le cas contraire, l'adresse correspondante sur support externe.

- amener la zone référencée en mémoire.
  Le seul avantage de cette méthode est le gain de place mémoire, gain inutile puisqu'il ne tire pas profit du concept des machines virtuelles.

2.3.1.4 Sous-environnement graphique d'analyse

Dans le sous-environnement d'analyse l'utilisateur peut activer un ensemble d'outils graphiques lorsque le terminal utilisé par XCP est un terminal graphique IBM 2250. Cet ensemble de requêtes est volontairement autonome car il fait intervenir des dispositifs particuliers à cette unité d'entrée/sortie.

2.3.2 Le support graphique

L'environnement XCP est un outil conversationnel et pour cela utilise comme moyen de communications homme-machine un terminal classique ou un terminal graphique IBM 2250. Cette possibilité de choix est imposée par le fait que l'unité 2250 n'existe qu'en un seul exemplaire à l'IMAG, et nous n'avons pas voulu que son immobilisation empêche le fonctionnement de XCP.

L'emploi d'une unité 2250 est à notre avis parfaitement justifié, du fait de la grande masse d'informations à examiner, et du caractère intéractif de XCP. L'écran cathodique permet un échange beaucoup plus rapide qu'avec les terminaux classiques et permet de présenter instantanément à l'utilisateur un sous-ensemble important des données qui l'intéressent : ainsi, un bloc de contrôle du système peut être affiché dans sa totalité et plus généralement n'importe quelle zone de mémoire peut être visualisée sans délai par rapport au terminal classique. De plus, le dialogue est encore amélioré par l'usage des touches de fonctions programmées et du pointeur optique. Avec des dispositifs une requête se traduit simplement par une pression sur une touche ou par une désignation sur l'écran. L'emploi du pointeur optique est particulièrement adapté à cette utilisation intéractive, car désigner un objet sur l'écran est un geste tout à fait naturel et exprime son intention sans ambiguité. L'unité 2250 a toutefois un seul inconvénient: c'est de ne laisser aucune trace des diverses manipulations. Pour y remédier nous avons défini une fonction "hard copy" qui reporte le contenu de l'écran sur une imprimante.

2.3.3 Exemple d'utilisation

```
* l'environnement XCP est activable sous le systeme CMS
* par la requete :

XCP ( type )
X.C.P READY ON 05/02/73 AT 18:15:15

DEFINITION ENVIRONMENT ...ENTER REQUEST

* l'option type est donnee pour indiquer que le terminal
* de dialogue est un terminal de type 'machine a ecrire'
* l'utilisateur a d'abord acces au sous-environnement de
* definition. Tous les descriptions de variables sous
*lues au moment de l'initialisation de XCP.

list sfblok
000 SFBLOK STRUC
001 SFBNEXT REF ( SFBLOK )
002 ADDR BYTE ( 7 )
003 DEVCODE BYTE ( 1 )
004 MRDEBLOK REF ( MRDEBLOK )
005 USERID CHAR ( 8 )
006 END

* la structure SFBLOK a ete definie lors d'une session
* precedente.
* la liste des requetes autorisees sont obtenues par :
request
CP          CVT         DEFINE     ENDEF      LIST       MODIFY     QUERY
SAVE        SET         SUPPRESS   STOP

cp query virtual


CORE - 01280K
009 CONSL 05D
00C SPOOL RDR
00D SPOOL PUN
00E SPOOL PRT
010 SPOOL RDR
0FF RPQ TIMER
190 2314- 230 054 CYL R/O CP0000
191 2314- 336 020 CYL R/W CPDS12



* la liste ci-dessus donne la configuration de la machine
* virtuelle sur laquelle fonctionne XCP. On remarque que
* la memoire est de 1280k = 256k (CMS) + 1024k (image memoire
* de CP-67 a analyser)
define runuser ref(utable    )     base(  #160)

*la requete define permet de definir les variables utilisees.


* l'utilisateur a defini  ici une variable simple qui sera
* rangee dans le fichier SYMBOLS CP-67
list runuser
RUNUSER REF ( UTABLE ) BASE ( #160 )
```

```
define utable struc
ALREADY EXISTS

define vdevblok struc  base(vdevlist)
ENTER STRUCTURE :
vdevpnt ref(vdevblok)
vdevadd byte(h)
vdevadd char(8)
ALREADY EXISTS
vdevstat bitst(1
INCORRECT SYNTAX, RETYPE
vdevstat bitst (1)
vdevtype byte(1)
vpntreal ref(rdevblok)
vdevrel byte(2) base (16)
vdevbnd byte(2)
vdevpos byte(8)
vdevsnse byte(6)
vdevflg bitst(1)
_  tempdev bit(7)
readonly bit(6)
vdevslen byte(1)
vdvdial bit(3)
vshared bit(5)
vde@venbl bit(4)
end

save vdevblok

* la requete precedente sauve la structure VDEVBLOK sur
* support externe
query core
READ FROM REMOTE-PRINTER

* par defaut l'image memoire a analyser est sur un
* fichier de "spooling"
set core disk

* cette requete indique a XCP que l'image memoire doit
* etre lue sur un disque.
* L'utilisateur signale qu'il veut entrer dans le sous-
* environnement de definition par la requete ENDEF avec
* l'option SKIP qui permet d'ignorer  les erreurs even-
* tuelles detectees lors de la generation des descripteurs
* internes correspondants aux definitions de variable.
end skip

UNRESOLVED REFERENCES:
SHASEG   IN EXTUTAB
VSEGTAB  IN EXTUTAB
SHASEG   IN EXTUTAB
MRIBUFF  IN MRDEBLOK
MVIBUFF  IN MVDEBLOK
ALLOCTAB IN RDEVBLOK
```

```
SHARESYS IN UTABLE

GIVE FILENAME AND FILETYPE OF DUMP

dump02  v3

DUMP OF CP-67 ON  3/01/73 AT 15.35.53
EXAM ENVIRONMENT ...ENTER REQUEST

* dump03 v3 sont le nom et le type du fichier contenant
* l'image memoire a analyser.
* Les erreurs listees sont des references non resolues
* entre les differentes variables definies.
req
ACTIVATE  BACK      CALL      CP        CVT       DISCARD   DISPLAY
DUMP      EXIT      FIND      FORGET    LIST      NEXT      QUALIFY
QUERY     REMEMBER  SEARCH    SET       SHOW      SNAP      STOP

* L'ensemble des requetes du sous-environnement d'analyse est liste .

&users
LISTE DES UTILISAT ACTIFS
ASSISTAN
SPEC02
MESURE
FORMAC
BATCH
TP3
SYS07
(LOGIN)
TP4
TP8
KONILEDU
APL360
ISN2
VMCPGEN
CMSMESUR
AUTOMA22
SYGESCO
GETA05
SOCRATE7
VIVIER
ILL03
ILL50
PASCAL
TEG01
LEVY
CTIP03
IFP
CIT08
ARCHIVES
ILL30
TP9
EBERHARD
CTIP02
THI
PLANNING
```

```
IMAG2
MASSILIA
OPERATOR
CNRSBT
ETALG02
TEG02
BENJOU
TP7

* &users est une macro-requete qui liste
* les noms de toutes les machines virtuelles actives
* lors de la panne de CP-67

list &users
000 &SET PROC OFF
001 &SET REQ OFF
002 QUALIFY UTABLE
003 SHOW USERID
004 &PRINT LISTE DES UTILISAT. ACTIFS
005 &FIRST = &OUT(13)
006 &PRINT &OUT(#D)
007 BOUC: NEXT 1
008 SHOW USERID
009 &IF &OUT(13)=&FIRST &RETURN
010 &PRINT &OUT(13)
011 &GOTO BOUC

* encore un exemple d'appel de macro-requete


&spool punchers
NOMBRE DE FICHIERS SPOOL = 0



&spool printers
LISTE DES USERS AYANT UN FICHIER EN SPOOL PRINTERS
LEVY
LEVY
ETALG02
SYGESCO
LEVY
CALLEJC
LEVY
BATCH
PEQUIGNO
SYGESCO
OPERATOR
KONILEDU
FORMAC
THI
SYGESCO
SYS07
SYS07
CMSMESUR
FORMAC
NOMBRE DE FICHIERS SPOOL = 19
```

```
query page
AVAILABLE CORE:
PAGE #000 TO #018
PAGE #0E2 TO #0FF


qualify utable

* activation de la structure UTABLE. L'occurrence qu'on
* accede est celle qui est pointee par RUNUSER ,prece-
* demment definie


show userid vpsw vmstatus
USERID      "ASSISTAN"
VPSW        000400CA50011000
VMSTATUS    '1000000000000110'
PAGEWAIT    '1'B
IOWAIT      '0'B
CFWAIT      '0'B
SYSOPBIT    '0'B
COMSW       '0'B
VIRCOMSW    '0'B
INLOGOFF    '0'B
INLOGON     '0'B

* la requete SHOW a demande l'impression des
* informations appartenant a la structure active
query acces
POSSIBLE ACCESS:
SEGTABLE->SEGTABLE
VCHSTART->VCHBLOK
NEXTUSER->UTABLE
VMXSTART->MVDEBLOK
ADEXTAB->EXTUTAB
NEXTRTMR->UTABLE
NXTQ->UTABLE

* les structures accessibles a partir de la
* structure active sont listees
qualify vchblok

* activation du bloc decrivant un canal virtuel
qualify vcublok


* activation du bloc decrivant une unite de controle
* virtuelle.

q vdevblok

next 1

query access
POSSIBLE ACCESS:
VDEVPNT->VDEVBLOK
VPNTREAL->RDEVBLOK
```

```
show vpntreal->all
RDEVPNT      00000000
RDEVCU       00012BD0
RDEVADD       0093
RDEVTYPE     84
RDECUPTH     40
RDEVTASK     00000000
RVOLSER      "CPDS17"
RDEVCODE      00FF
RDEVALLN     00000000
RDEVERCT     +0
RDEVSTAT     '0000001000000000'
RDEVOWND     '0'B
RDEVATTD     '0'B
RDEVDED      '0'B
RDEVSEEK     '0'B
RDEVPOSD     '0'B
RDEVSYS      '1'B
RDEVUSER     00000000
RATTVADD      0000
RDEVFTR      00
RDEVSLEN     00
RDEVSEN       0000 0000 0000
RDEVCXX      0000000000
RDEVTMON     0000000096

* la requete ci-dessus donne le contenu
* de RDEVBLOK qui est le bloc de controle du
* disque CPDS17 servant a simuler le disque 193 virtuel
remember actif

* le chemin parcouru depuis la premiere requete QUALIFY
* est memorise

discard until utable
UTABLE   IS ACTIVE

* desactivation de toutes les structures, sauf la premiere
search iowait='1'
CONDITION SATISFIED

* SEARCH a permi s de selectionner le premier utilisateur
* ayant le bit entree/sortie en attente egal a 1
show userid pending vmstatus vpsw
USERID       "FORMAC  "
PENDING      '0000000000000000'
VMSTATUS     '0100000000000110'
PAGEWAIT     '0'B
IOWAIT       '1'B
CFWAIT       '0'B
SYSOPBIT     '0'B
COMSW        '0'B
VIRCOMSW     '0'B
INLOGOFF     '0'B
INLOGON      '0'B
VPSW         0004000090003770
```

```
show vchstart->vculist->vdevlist->vpntreal->rdevtask
RDEVTASK     00000000


discard all

qualify rchblok

next 1

show all
RCHANPNT     00012DD0
RCULIST      00012AB0
TASKLIST     00000000
RCUACT       '01111111'
RCHSTAT      '11000000'
BUSY         '1'B
RESCAN       '1'B
RCUCOUNT     +1
RCHANADD      0100
TASKCNT      +0
TAKLAST      00012AD0
RCHCOND       0000
RCHDATCK     00
RCHCONCK     00
RCHIFCC      00
RCHANCC      00
** FROM 012AEE TO 012AEF =FILLER **

n


que access
POSSIBLE ACCESS:
RCHANPNT->RCHBLOK
RCULIST->RCUBLOK
TASKLIST->IOTASK
TAKLAST->IOTASK

q iotask

show iotask
TASKRDEV     00012AF0
TASKRCU      00012BD0
TASKPNT      00000000
TASKPATH     '01000000'
TASKFLAG     '00011000'
IOERROR      '0'B
CPIOERR      '1'B
CPSPSEEK     '1'B
CHANFREE     '0'B
PROCC1       '0'B
ALONSEEK     '0'B
TASKADD       FF00
TASKUSER     000FE8C0
TASKCAW       000F07B0
TASKIRA       0000C0F8
```

```
display #f07b0 ccw(8)@@5)
#0F07B0:      #07,#0F07D8,SILI,#0006
              #31,#0F07DA,CD,#0005
              #08,#0F07B8,0,#0000
              #05,#0F07E2,CD+SILI,#033D
              #03,#000000,SILI,#0001

disp #f07d8 byte((3)h)
#0F07D8:      0000 00AB 000B

qual taskuser->utable


sh userid vpsw vmstatus
USERID       "IMAG2   "
VPSW         0000000080000C0C4
VMSTATUS     '0100000000000110'
PAGEWAIT     '0'B
IOWAIT       '1'B
CFWAIT       '0'B
SYSOPBIT     '0'B
COMSW        '0'B
VIRCOMSW     '0'B
INLOGOFF     '0'B
INLOGON      '0'B

activate actif

* activation d'un chemin memorise precedemment; le
* chemin actif est desactive implicitement par ACTIVATE


query trace
UTABLE   WAS QUALIFIED
#0FD458
VCHBLOK  WAS QUALIFIED
#0E83D8
VCUBLOK  WAS QUALIFIED
#0EABB8
VDEVBLOK WAS QUALIFIED
#0ECF48

qu vpntreal -> rdevblok
next 1;q@sh rvolser

RVOLSER      "CPDSK2"

qualify rdevcu->rcublok

qual ractchan->rchblok

q iotask

qua taskuser->utable

show userid
USERID       "IMAG2   "

stop
R; T=1.84/5.04 18:44:03
```

C H A P I T R E 3

XCP VU PAR L'UTILISATEUR

CHAPITRE 3

L'application de XCP au système CP-67 (et éventuellement à tout autre système d'exploitation) comporte 2 phases distinctes :

1/ La description des structures des données du système à analyser

Cette phase permet de décrire à l'aide du langage spécialisé de XCP, les caractéristiques de chaque donnée (type, longueur, nombre d'éléments, etc...) et les relations pouvant exister entre certaines de ces données.

En effet, comme tous les systèmes d'exploitation, CP-67 utilise des données complexes telles que des structures, des tableaux, des pointeurs, etc...(figure 3.1). Il est difficile pour un être humain de démêler cet enchevêtrement de blocs; aussi devant cette complexité, il est souhaitable de libérer l'utilisateur des tâches les plus rébarbatives afin qu'il se consacre uniquement à son problème de mise au point. Ce souhait ne peut se réaliser que si l'outil fournit une description non ambigüe de la structure des données. Or, actuellement la description des données de CP-67 est faite en langage d'assembleur 360 et ne donne que très peu de renseignements utiles. Par exemple, le pointeur sur la structure UTABLE est défini par :

```
RUNUSER     DS      F.
```

D'après cette définition, rien ne nous renseigne sur :

- sa nature (pointeur)
- son adresse (emplacement mémoire)
- ses relations éventuelles avec d'autres données

Il en est de même pour toutes les autres variables contenues dans les programmes constitutifs de CP-67.

2/ L'exploitation de la précédente description

pour explorer, analyser une image du système observé, au moyen des mécanismes de base fournis par XCP.

3.1 DESCRIPTION DES DONNEES

Dans XCP, toute utilisation d'une variable symbolique doit faire l'objet d'une déclaration. Elle est faite de façon intéractive pour permettre à l'utilisateur de corriger au fur et à mesure les fautes de syntaxe. Les déclarations peuvent être soit temporaires (valable uniquement pour la session) soit permanentes (stockées sur support secondaires, pour être utilisées dans les sessions ultérieures).

3.1.1 Langage de description

Pour donner les caractéristiques des variables qu'il veut manipuler, l'utilisateur dispose d'un langage de description de données analogue aux langages évolués (§ PL/1, Algol W etc...). La syntaxe utilisée a conservé la forme de ces langages pour deux raisons :

- ne pas rebuter l'utilisateur par l'apprentissage d'un langage entièrement nouveau.
- être simple de manière à permettre une réalisation rapide et commode.

La syntaxe du langage de description est définie par l'ensemble des règles de grammaire suivantes :

<déclaration> ::= <caractéristiques>|<caractéristiques><base>

<caractéristiques> ::= <nom> <type>|*(<longueur1>)|

<nom> struc<liste composants> end

<liste composants> ::= <caractéristiques> ; <liste composants>|

<caractéristiques>

<base> ::= base (<adresse>)

<adresse> ::= <adresse symbolique>|<adresse absolue>

<adresse symbolique> ::= <identificateur>

<adresse absolue> ::= <nombre>

<nom> ::= <identificateur>|<identificateur>(<nb d'éléments>)

<nb d'éléments> ::= <nombre>

<type> ::= <type1>(<longueur1>)|<type2>(<longueur2>)|

<type3>(<référence>)|<type4>(<numéro bit>)

<type1> ::= bitst | byte | char | dec | int

<type2> ::= bitst | field

<type3> ::= ref | struc

<type4> ::= bit

<longueur1> ::= <longueur numérique>|<longueur symbolique>

<longueur numérique> ::= <nombre>

<longueur symbolique> ::=(<facteur duplicatif>) <unité mémoire>

<facteur duplicatif> ::= <nombre>

<unité mémoire> ::= d | w | h

<longueur2> ::= <borne inférieure> : <borne supérieure>

<borne inférieure> ::= <nombre>

<borne supérieure> ::= <nombre>

<référence> ::= <identificateur>

<numéro bit> ::= <nombre>

<nombre> ::= <nombre décimal>|<nombre hexadécimal>

<nombre décimal> ::= <chiffre décimal>|<chiffre décimal><nombre décimal>

<chiffre décimal> ::= 0 | 1 | 2 | ..... | 9

<nombre hexadécimal> ::= # <partie numérique>

<partie numérique> ::= <chiffre hexadécimal>|<chiffre hexadécimal><partie numérique>

<chiffre hexadécimal> ::= <chiffre décimal> |A|B|....|F

<identificateur> ::= <lettre>| <identificateur><caractère alphanumérique>

<caractère alphanumérique> ::= <lettre>|<chiffre décimal>

<lettre> ::= A|B|C|....|Y|Z

D'après les règles précédentes, la description d'une variable peut se résumer à donner des informations concernant :

- son identification
- son type
- sa base

3.1.1.1 Identification

C'est ce qui permet d'identifier la variable déclarée. Cet identificateur peut être indicé; dans ce cas la variable possède plusieurs éléments (tableau). Seuls sont autorisés les tableaux à une dimension, avec indice de 0 à n-1.

EXEMPLES :

RUNUSER	variable non indicé
VGPRS(16)	tableau de 16 éléments

REMARQUES

- Le nombre maximal d'éléments dans un tableau est de 65536.
- si une variable a n éléments, ceux-ci sont alors référencés par des indices de 0 à n-1
- la longueur maximale d'un identificateur est en pratique limitée à 8 caractères.

3.1.1.2 Type

Il constitue la partie principale de la description; il précise la nature et la longueur de la variable. Dans certains cas, les renseignements sur la longueur peuvent être remplacés par une référence qui est une information de lien entre 2 variables de type pointeur ou de type structure.

<type> ::= <type1>(<longueur1>)|<type3>(<référence>)

3.1.1.2.1 Les_différents_types

Il existe 7 types de variables dont l'existence est liée à la nature des informations manipulées et au système 360. Ces types sont :

BITST
BYTE
CHAR
DEC
INT
REF
STRUC

Chaque fois qu'une variable est référencée (implicitement ou explicitement) son contenu est vérifié en fonction du type déclaré.

L'utilisateur est averti par un message d'erreur lorsqu'il n'y a pas conformité entre le type déclaré et le type de la donnée manipulée.

a) BITST permet de définir des variables représentant une chaîne de bits, c'est-à-dire une suite de 0 et de 1.

b) BYTE représente une donnée de type hexadécimal

c) CHAR permet de définir une chaîne de caractères

d) DEC permet de définir des données en code décimal condensé [Ref.11].

e) INT représente une donnée de type entier virgule fixe.

f) REF permet de définir une donnée contenant une information de lien, c'est-à-dire l'adresse d'une autre variable (pointeur).

g) STRUC: Toutes les variables citées précédemment sont dites variables simples, par opposition aux variables structurées. Par définition, une variable de type STRUC est un ensemble de variables simples, ou structurées. Le début d'une variable structurée est signalée par l'en-tête spécifiant le type STRUC et la fin de la structure est signalée par END.

REMARQUE

- Niveaux: une définition de structure peut inclure d'autres définitions de structure et récursivement. Par définition la structure principale possède le niveau 1 tandis que les structures internes ont le niveau de la structure qui l'entoure plus 1.

```
Niveau 1        A _ STRUC     ...
                  =
                  =
                  -

Niveau 2        B _ STRUC     ...
                  =
                  =
                  =
                  -

Fin niveau 2    END
                  =
                  =
                  -

Niveau 2        C _ STRUC
                  =
                  =
                  -

Fin niveau 2    END
                  =
                  -

Fin niveau 1    END
```

- Variable de remplissage ("filler"): Dans une variable structurée, certaines parties de la structure peuvent être ignorées en leur donnant comme identificateur de variable le symbole *. Une telle variable * n'a pas de type spécifique puisqu'elle n'a pas de signification pour l'utilisateur.

3.1.1.2.2 Longueur ou référence

Sauf pour les variables de type REF et STRUC, la longueur doit être spécifiée :

- soit en nombre d'octets (longueur numérique)
- soit en nombre de demi-mots, mots ou double-mots (longueur symbolique).

a) Longueur numérique

Pour uniformiser les définitions de variables, nous avons pris comme longueur de référence l'octet c'est-à-dire l'unité d'adressage du système 360. La longueur d'une variable est précisée en donnant simplement le nombre d'octets consécutifs occupés.

EXEMPLES

BITST (3)
Chaîne de bit ayant une longueur de 3 octets

CHAR (8)
Chaîne de caractères ayant une longueur de 8 octets

REMARQUE

Les variables de type INT ne peuvent avoir comme longueur que 2, 4 ou 8 octets du fait de l'architecture du système 360.

b) Longueur symbolique

Une longueur symbolique est définie par les mots-clés H, W, D qui désignent respectivement des demi-mots, des mots et des double-mots.

EXEMPLES

INT (H)
entier de longueur 2 dont l'adresse est alignée sur une frontière de demi-mot.

Au cas où une variable possède une longueur multiple d'une longueur symbolique, un facteur de répétition peut être spécifié.

EXEMPLES

- BYTE ((3)H)
  Variable hexadécimale de longueur 12 octets et dont l'adresse est alignée sur une frontière de demi-mot.

- CHAR ((2) D)
  Variable chaîne de caractères de longueur 16 octets et dont l'adresse est alignée sur une frontière de double mots.

REMARQUE

Les variables de type INT ne peuvent pas avoir un facteur de répétition à cause des contraintes sur les longueurs.(§ remarque précédente).

c) Référence

Lorsqu'une variable est de type REF ou STRUC, une "référence" est utilisée à la place de la longueur, car cette dernière est connue implicitement. En effet, un pointeur dans le système 360, a une longueur de 4 octets, tandis que la longueur d'une variable structurée est par définition la somme des longueurs des variables simples qui la compose.

- "référence" pour une variable de type REF, doit être l'identificateur de la variable adressée (variable pointée)

EXEMPLE

Si la variable RUNUSER contient l'adresse de la variable UTABL elle est alors déclarée :

```
RUNUSER  REF(UTABLE)
```

- "référence" pour une variable STRUC, ne doit être utilisée que si la structure que l'on veut définir possède les mêmes éléments qu'une autr structure. "référence" désigne alors l'identificateur de cette dernièr

EXEMPLE

La structure SVCOPSW possède les mêmes éléments que la structu SVCNPSW. On peut définir simplement la structure de la façon suivante :

```
SVCOPSW  STRUC (SVCNPSW)
```

On dit alors que la structure en cours de définition est une copie de la structure nommée en "référence".
Cette façon de procéder est pratique lorsqu'il existe des variables de noms différents mais structurées de façon identique car elle permet de réduire la longueur des définitions.

REMARQUES

- "référence" n'est utilisé que pour les copies dans les variables de type STRUC. Autrement la partie (<référence>) est absente.

- pour des raisons de réalisation, une structure copiée ne peut être qu'une structure de niveau 1, ne comportant aucune sous-structure, donc à fortiori des références à des copies.

3.1.1.2.3 Cas particulier des variables BITST et BYTE

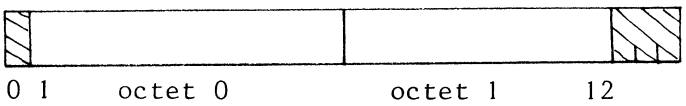
a) BITST

Certaines chaînes de bits n'ont pas toujours comme longueur un multiple entier d'octets, mais occupent consécutivement un certain nombre de bits.

EXEMPLE

```
A     BITST (1:12)
```

La variable A est une chaîne de bits, dont les limites sont le bit numéro 1 et le bit numéro 12 (les bits sont numérotés



de gauche à droite, le premier ayant le numéro 0.

Dans une chaîne de bits, on peut individualiser certains d'entre eux en leur donnant un identificateur.

EXEMPLE

Dans la chaîne de bits VMSTATUS du système CP-67, le bit numéro 5 indique si une machine virtuelle est en attente de page. On peut avoir :

```
VMSTATUS      BITST (2)
PAGEWAIT      BIT   (5)
```

REMARQUES

- Les définitions de bits ne peuvent apparaître qu'après une définition de chaîne de bits.
- Les bits peuvent être définis dans n'importe quel ordre pourvu que leur numéro soit compris dans la chaîne.
- Un bit peut avoir plusieurs identificateurs possibles.
- Une variable autre que BIT termine la liste de définition des bits dans une chaîne de bits.

b) BYTE

Une variable de type BYTE peut être structurée en plusieurs champs, comme une chaîne de bits l'est en plusieurs bits. Ceci est possible par la définition de champ FIELD.

EXEMPLE

La variable MASK est partagée en 3 champs
- ILC (bits 0 à 1)
- CC (bits 2 à 3)
- PGMASK (bits 4 à 7)

et est déclarée par :

```
MASK   BYTE(1)
ILC    FIELD(0:1)
CC     FIELD(2:3)
PGMASK FIELD (4:7)
```

REMARQUES

- Les déclarations FIELD ne peuvent apparaître qu'après celle d'une variable de type BYTE.
- Les différents champs peuvent apparaître dans n'importe quel ordre

- Les bornes d'un champ ne doivent pas dépasser les spécifications de longueur de la variable BYTE.

- Une déclaration autre que FIELD termine la structuration de la variable BYTE.

3.1.1.3 Base

La base (ou adresse) d'une variable doit être spécifiée dans la dernière partie de la définition par

BASE(adresse absolue | adresse symbolique)

La base est obligatoire pour les variables simples, tandis que pour les variables structurées, elle peut n'être spécifiée qu'au moment de l'utilisation de la structure (§3.2.1). Certaines variables remplissant des fonctions bien précises ont leur base connues de XCP. Dans les définitions de ces variables, la partie "base" peut être omise, et la base sera automatiquement affectée, lorsque ces variables seront rencontrées.

Exemple

L'horloge est un entier de longueur d'un mot situé à l'adresse 128. Elle peut être déclarée par

```
TIMER  INT (W)
```

ou

```
TIMER INT(W)    BASE(128)
```

Remarque

Les variables citées ont des emplacements fixes définis par l'architecture du 360. Elles ne peuvent être reconnues que par des identificateurs prédéterminés. Par exemple :

```
TIMER   horloge
SVCNPSW nouveau PSW interruption SVC
SVCOPSW ancien PSW interruption SVC
etc...
```

Pour les variables dont la base est connue, celle-ci peut être exprimée soit de façon absolue, soit de façon symbolique.

a) Adresse absolue

La base de la variable est donnée par un nombre désignant un emplacement en mémoire.

b) Adresse symbolique

La base de la variable est donnée par l'identificateur de la variable de type REF qui contient l'adresse de la variable en cours de définition (réciproque de l'attribut référence).

Exemple

```
SFBLOK STRUC  BASE (APRINTER)
```

l'adresse de SFBLOK est contenue dans la variable APRINTER.

Remarques

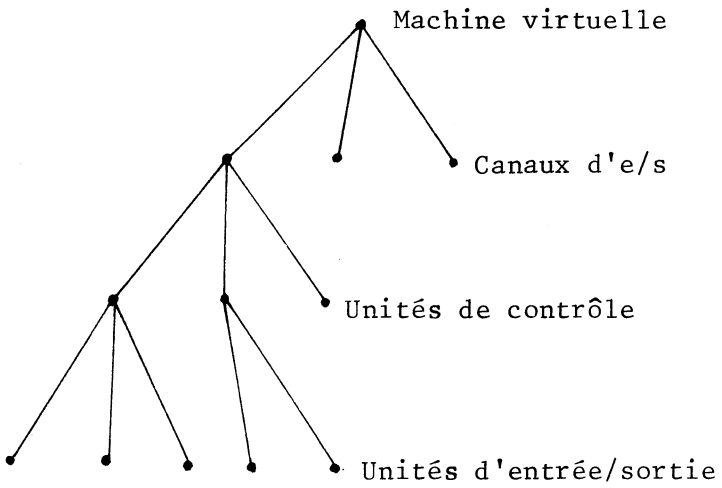
- Le type de la variable spécifié dans BASE doit être de type REF.
- La base d'une variable spécifiée dans BASE peut être une adresse symbolique et ceci peut se répéter. Autrement dit l'adresse d'une variable peut être une série de pointeurs, pourvu que la base de la dernière variable REF soit une adresse absolue.

Exemple :

```
A   REF(B)      BASE(4)
B   REF(C)      BASE(A)
C   REF(D)      BASE(B)
D   BYTE(8)     BASE(C)
```

3.1.2 Exemple de description

La structure suivante définit dans CP-67 un canal virtuel pour une machine virtuelle. Chaque canal possède plusieurs unités de contrôles, et à chaque unité de contrôle on peut connecter plusieurs unités virtuelles.




```
VCHBLØCK  STRUC  BASE(VCHSTART)
VCHANPNT  REF   (VCHBLØK)
VCULIST   REF   (VCUBLØK)
VCHANADD  BYTE (H)
VCUCØUNT INT (H)
VCHSTAT   BITST (1)
BUSY   BIT (3)
CEBIT (4)
* (3)
VCEUNIT   BYTE (H)
VNPNDCUI   INT (H)
* (W)
VCHCSW   STRUC (CSW)
END
```

VCHSTART est la base d'un bloc VCHBLOK. Cette base se trouve dans la UTABLE. Dans cette structure, il existe 2 pointeurs comme le montre le schéma précédent :

1 sur le prochain VCHBLOK (pointeur VCHANPNT)
1 sur la liste des VCUBLOK (pointeur VCULIST)

VCHSTAT est une chaîne de bits dans laquelle, on a défini les bits numéro 3 et 4 respectivement comme BUSY et CE.

Enfin dans la structure VCHBLOK, on trouve une structure de niveau 2, VCHCSW qui est la réplique exacte de CSW, mot d'état d'un canal.

3.1.3 Fichiers symboliques

Les définitions de variables peuvent être rendues permanentes d'une session sur l'autre en conservant leurs descriptions dans des fichiers qui sont lus automatiquement à chaque initialisation de XCP.

Les variables simples sont rangées dans le fichier SYMBOLS CP-67, tandis que les variables structurées sont stockées séparément dans des fichiers de même nom que l'identificateur de la structure.

3.2 ANALYSE D'UN INSTANTANE

Pour son travail de diagnostic et de recherche des erreurs, l'utilisateur dispose d'un ensemble de requêtes permettant de dialoguer avec l'environnement XCP. Ces requêtes se divisent en 2 sous-ensembles :

1/ Sous-environnement de définition

Sous-ensemble de requêtes permettant de travailler sur les déclarations des variables symboliques (définition, correction, examen, etc...)

2/ Sous-environnement d'analyse

Sous-ensemble des requêtes permettant le dépouillement d'une image mémoire ou d'un instantané de CP-67.

Ces deux sous-ensembles possèdent une partie commune qui forme le sous-ensemble des requêtes de service.

3.2.1 Les requêtes de base

Une requête de base correspond à une action ou une fonction déterminée de XCP. Cette action varie avec les paramètres d'utilisation de la requête, mais ne peut être changée dans sa forme globale, ceci par opposition aux macro-requêtes (§ 3.2.2) dont les actions sont modifiables au grè de l'utilisateur. Le nombre de requêtes de base est fixe alors que celui des macro-requêtes peut être quelconque et est fonction des besoins.

3.2.1.1 Les caractères spéciaux

Certains caractères sont utilisés soit comme séparateurs, soit comme préfixes ou encore comme opérateurs dans les requêtes.

a) Séparateurs

1) Le caractère blanc sépare les différents paramètres dans une requête et de ce fait peut apparaître en nombre quelconque. Il n'est significatif que dans une constante de type 'chaîne de caractères' (voir plus loin).

2) Le caractère ; indique la fin logique d'une ligne d'entrée.

3) La paire de parenthèses délimite, soit l'indice d'un élément dans un tableau, soit une référence.
VGPRS(0) est le premier élément du tableau VGPRS.

b) Préfixes

1) Le caractère # sert de préfixe aux nombres hexadécimaux. Dans toutes les requêtes les nombres peuvent être exprimés indifféremment en base 10 ou en base 16.

Exemple :

#A représente le nombre 10 en hexadécimal

2) Le caractère "encadre une chaîne de caractères dans laquelle est exclue le caractère " lui-même.

Exemple :

"ABC" représente les caractères ABC

3) Le caractère ' encadre une chaîne de bits, c'est-à-dire une suite de chiffres 0 ou 1.

Exemple :

'10010' est la chaîne de bits 10010

4) Le caractère ε sert de préfixe aux noms des requêtes et aux mots clés du langage d'extension. (§ 3.2.2).

Exemple :

εSET est le nom d'une requête du langage d'extension

c) *Opérateurs arithmétiques et logiques*

1) Les caractères + - * / sont les opérateurs arithmétiques classiques sur les nombres entiers. (La multiplication et la division ont la plus grande priorité).

2) Les opérations logiques sont représentées par les caractères (ou combinaisons de caractères) suivants :
< > <= >= = ¬=

3) La combinaison des 2 caractères -> est utilisée avec les variables de type REF.
A → B signifie que la variable A se refère à la variable B. A doit être obligatoirement de type REF.
L'opérateur -> peut se répéter si la variable référencée est une variable de type REF.
A -> B -> C -> D signifie que la variable D est référencée par une chaîne de variables REF A, B et C.
Les opérandes de l'opérateur -> peuvent être des nombres, à condition que leurs valeurs soient des multiples de 4. Ceci est dû au fait que les constantes adresses doivent être contenues dans un mot machine dans le système 360.

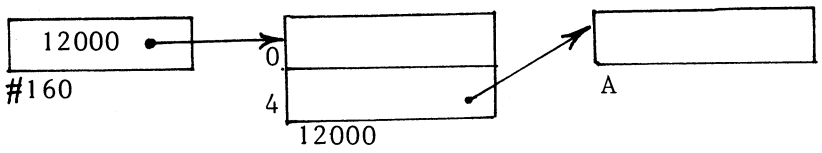
Remarque :

> Lorsque les opérandes sont des nombres, le premier nombre est toujours une adresse absolue, tandis que les autres nombres sont des adresses relatives.

Exemples :

- #160 -> A signifie que l'adresse de la variable A est contenue dans la mémoire d'adresse absolue hexadécimale #160.

- #160 -> 4 -> A signifie que l'adresse de la variable A est contenue dans le mot d'adresse relative (offset) 4, à partir de l'adresse contenue dans la mémoire d'adresse absolue #160.



4) Enfin pour les variables ou constantes de type chaîne de caractères, le caractère | est l'opérateur de concaténation.

3.1.1.2 Forme des requêtes

Une requête de base peut être formalisée par :

<requête> ::= <nom><paramètres>

<nom> ::= ACTIVATE|DEFINE etc...

<paramètres> ::= nom de variables ou valeurs|<mots clés>

<mots clés> ::= USING|AT etc...

Remarques

- les noms des requêtes ou les mots-clés peuvent être donnés en abrégé dans la mesure où il n'existe pas d'ambiguité possible.
- Les valeurs des paramètres dépendent du contexte et de la manière dont on veut utiliser des requêtes.
- Les mots clés ont seulement pour rôle de favoriser la compréhension des requêtes dans lesquelles ils apparaissent.

3.2.1.3 Les différentes requêtes de XCP

Nous mentionnons ci-après la liste des requêtes des deux principaux sous-environnements de XCP. Les détails concernant la format de chacune de ces requêtes sont donnés dans l'annexe de cet ouvrage.

a) Requêtes du sous-environnement de définition

CP
CVT
DEFINE
ENDEF
LIST
MODIFY
QUERY
SAVE
SET
SUPPRESS
STOP

b) Requêtes du sous-environnement d'analyse

ACTIVATE
BACK
CALL
CP
CVT
DISCARD
DISPLAY
DUMP
EXIT
FIND
FORGET
LIST
NEXT
QUALIFY
QUERY
REMEMBER
SEARCH
SET
SHOW
SNAP
STOP

A tout moment la liste des requêtes du sous-environnement actif peut être obtenue par la frappe de REQUESTS.

3.2.1.3.1 Requêtes du sous-environnement de définition

a) DEFINE

Au moyen de la requête DEFINE l'utilisateur peut donner les définitions des variables qu'il veut étudier. Les paramètres de la requête DEFINE sont :

- soit une définition de variable simple
- soit la définition d'une en-tête de variable structurée.

Dans ce dernier cas, la fin de la définition doit être matérialisée par END.

Exemples :

- define runuser ref (utable) base (#160)
  définition de la variable simple RUNUSER.

- define a struc base (b)
  ENTER STRUCTURE:
  c ref (d)
  e struc
  f bitst (2)
  g char (8)
  end fin niveau 2
  h ref (z)
  end fin niveau 1

Les noms des variables simples structurées doivent être choisies de façon unique. Mais ceci ne s'applique pas aux variables internes à chaque structure de niveau 2 ou plus, car pour accéder à ces dernières,le nom de la structure principale doit être spécifié.

Une analyse syntaxique est faite sur chaque définition ou, pour une variable structurée, sur chaque définition d'une variable interne. Aucune vérification sémantique n'est effectuée puisque cette dernière opération n'est réalisée que lors de la génération des descripteurs internes (§ requête ENDEF).

La définition d'une variable est conservée en mémoire centrale, chaînée avec les définitions antérieures qui sont obtenues soit par des requêtes DEFINE, soit à partir des fichiers symboliques. Les

variables déclarées ne sont accessibles qu'au cours d'une session sauf si l'utilisateur les rend permanentes par la requête SAVE.

b) SAVE

Les définitions des variables peuvent être conservées sur support externe au moyen de la requête SAVE. Les variables simples sont alors rangées dans le fichier SYMBOLS CP-67 tandis que les variables structures sont rangées séparément dans les fichiers de type STRUC.

```
define   A   STRUC ....
-
-
-
end

save A          la variable A est rangée dans le fichier A de type
                STRUC
```

Lors de l'initialisation de XCP les variables permanentes sont automatiquement prises en compte à partir du fichier SYMBOLS et des fichiers de type STRUC.

c) Modify

Les structures déclarées par la requête DEFINE ne peuvent être corrigées que par la requête MODIFY , car chacune d'elles doit être vérifiée par l'analyse syntaxique avant son acceptation. La requête 'MODIFY' est en fait un mini-éditeur, puisqu'elle offre la plupart des opérations classiques d'un éditeur [Ref.41].

Chaque élément d'une structure est défini par son numéro d'ordre par rapport à l'en-tête de la structure. La correction ne se fait qu'au niveau de l'élément et les déplacements à l'intérieur de la structure sont spécifiés par le numéro correspondant. Les fonctions de cet éditeur sont :

- delete n — suppression de l'élément courant ou de n éléments à partir de l'élément courant.

- goto n — repérage d'un élément donné.

- print n — impression de l'élément courant ou de n éléments

- replace — remplacement de l'élément courant par l'élément donné en paramètre.

- lineno — valeur du numéro de l'élément courant.

- quit — fin de la requête MODIFY sans altération de la structure.

- file — fin de la requête MODIFY et prise en compte de toutes les modifications

d) Suppress

Les structures définies peuvent être détruites au moyen de cette requête. Si le paramètre FILE est présent dans la requête et si le fichier de type STRUC correspondant au nom de la structure existe, il est également supprimé.

e) List

La requête LIST est employée lorsque l'utilisateur veut obtenir des renseignements sur la définition d'une variable symbolique, par exemple l'impression d'une variable nommée, l'impression de toutes les définitions de variables d'un type donné, etc...

f) Endef

L'utilisateur émet cette requête lorsqu'il veut quitter le

sous-environnement d'analyse proprement dit.

Cette requête déclenche une série d'opérations nécessaires pour permettre l'examen symbolique d'une image mémoire de CP-67. Diverses options sont prévues pour que l'utilisateur puisse contrôler le déroulement de ces opérations :

- sauvegarde des définitions symboliques sur support externe, pour rendre ces variables permanentes.

- génération des descripteurs internes nécessaires au fonctionnement de XCP. Cette génération consiste en une compilation des déclarations des variables.

- résolution des références entre les descripteurs générés

- lecture du fichier contenant l'image mémoire de CP-67 si cette dernière est sur un support externe (§ requête SET).

Au cours de la compilation et de l'édition des références, certaines erreurs sémantiques peuvent être détectées et signalées à l'utilisateur. Par exemple :

- erreur d'alignement à l'intérieur d'une variable structurée

- référence à une variable non définie.

- numéro d'un bit hors des limites d'une chaîne de bits.

etc...

La détection d'une erreur force l'utilisateur à rester dans le sous-environnement de définition afin d'effectuer une correction immédiate. Diverses options de la requête ENDEF sont possibles pour :

- ignorer les erreurs détectées et rentrer inconditionnellement dans le sous-environnement d'analyse (option SKIP)

- supprimer la compilation des définitions de variables au cas où l'utilisateur n'emploie pas de variables (option NOGEN).

- forcer une sauvegarde sur support externe de toutes les structures déclarées (option SAVE).

3.2.1.3.2 Requêtes du sous-environnement d'analyse

a) Snap

Le mécanisme d'obtention d'une image mémoire de CP-67 se fait de deux manières différentes, soit par CP-67 lui-même à la suite d'erreurs détectées par les modules du système (§ chapitre 2), soit commandé par l'utilisateur à partir de sa machine virtuelle au moyen de la requête SNAP lorsque des conditions anormales se produisent.

Cette requête permet en effet de prendre un instantané de la mémoire du 360/67, sans pour autant perturber la marche du système. Le mécanisme déclenché par la requête SNAP est détaillé au chapitre 4 .

b) Qualify

A cause de la complexité des relations entre les différentes données et pour éviter les ambiguïtés entre les chemins possibles, l'accès à une structure doit être explicitement spécifié à XCP. L'exemple illustré par la figure 3.2 montre de telles ambiguïtés.

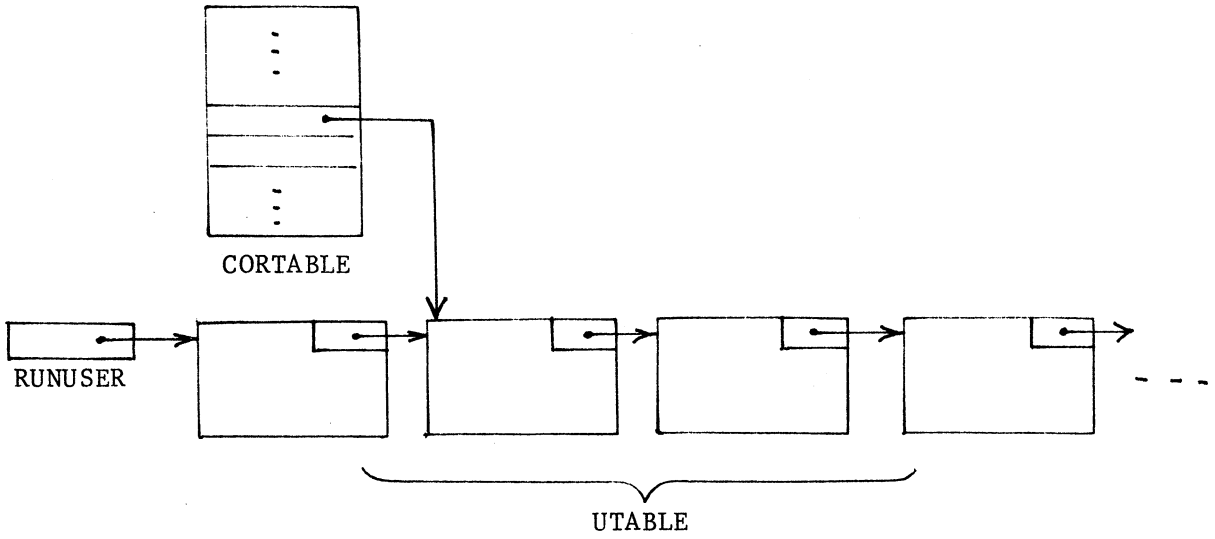


Figure 3.2

Chemins d'accès à une UTABLE

Dans CP-67 chaque machine virtuelle est décrite en mémoire par une table appelée UTABLE. Toutes les UTABLE sont chainées et la première est repérée par la constante-système RUNUSER. En principe on accède aux UTABLE par cette constante. Mais il existe également une autre possibilité qui est de prendre le pointeur sur la UTABLE située dans la CORTABLE, table donnant l'occupation de la mémoire centrale. De telles ambiguités d'accès se répètent souvent dans les données manipulées par CP-67.

Pour éviter ces inconvénients nous avons imposé à l'utilisateur d'expliciter les relations possibles entre les données (variable de type REF et adresse symbolique). De plus, à un instant précis, seules les variables d'une structure "qualifiée" sont accessibles. La requête QUALIFY sert à rendre accessible une structure désirée, à condition que cette structure soit repérée par une variable elle aussi accessible à l'utilisateur.

A l'initialisation de XCP, seules variables simples sont accessibles , car par leur définition même, elles n'ont pas besoin d'être "qualifiées". Les structures référencées par ces variables sont donc accessibles par la requête qualify et le processus recommence pour les structures référencées par des variables appartenant à la structure "qualifiée".

Exemple :

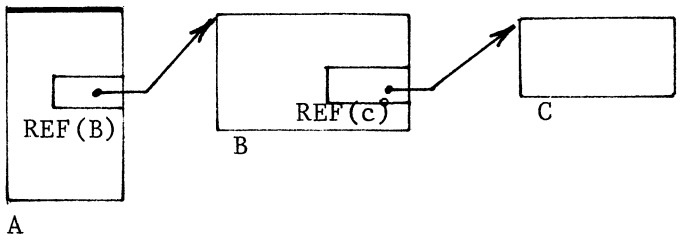


Figure 3.3

Relations entre les structures

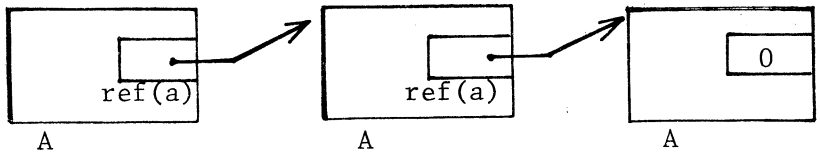
La figure 3.3 indique les relations existantes entre les 3 structures A, B et C et montre que pour accéder aux variables de la structure C, il faut accéder successivement et dans l'ordre à la structure A puis à la structure B. L'accès a C se fait donc par :

```
qualify A
qualify B
qualify C
```

En effet, pour atteindre le pointeur de B, la structure A doit être "qualifiée" et de même pour accéder à la variable REF(C) (pointeur de C) contenu dans la structure B il faut également "qualifier" la structure B.

c) Next

Lorsque la structure "qualifiée" est un élément d'une liste ou d'un anneau, c'est-à-dire qu'une telle structure possède une variable de type REF dont l'argument est le nom de la structure elle-même. L'accès à une occurrence quelconque de cette structure se fait par la requête next, qui permet de progresser dans une liste ou dans un anneau.



Pour accéder à la 3ème occurrence de la structure A, les requêtes suivantes sont nécessaires :

QUALIFY A	La structure A est "qualifiée" et sa première occurrence est active.
NEXT 2	Avancer de 2 éléments.

d) Back

Cette requête permet d'exécuter une "marche arrière", exactement à l'inverse de la requête NEXT, et sous les mêmes conditions.

Exemple :

Pour rendre active la deuxième occurrence de A après les 2 précédentes requêtes, il suffit d'émettre la requête BACK 1.

e) Discard

La requête DISCARD est utilisée pour désactiver une structure

"qualifiée" et réactiver en même temps une structure précédemment active. En effet, si nous avons successivement :

```
Qualify A
Qualify B
```

Après la requête QUALIFY B, l'utilisateur n'a plus accès aux variables de A. La requête DISCARD doit être alors utilisée pour désactiver B et réactiver A.

```
Qualify  A
|                       accès aux
|
|                       variables de A

Qualify B
|                       accès aux
|                       variables de B
Discard
|                       accès de nouveau
|                       aux variables de A
Discard                 aucun accès à A ou B
|
|
|
```

Le mécanisme des requêtes QUALIFY et DISCARD peut être comparé à celui d'une pile : "qualifier" une structure revient à la placer au sommet d'une pile d'où son accessibilité.
Désactiver une structure par DISCARD revient à l'enlever du sommet de la pile, ce qui a pour conséquences :

- rendre la structure inaccessible
- activer la structure précédemment "empilée".

f) Remember

Dans son travail d'investigation, l'utilisateur de XCP a

besoin de conserver le contenu de certaines variables, le chemin pris pour accéder à une structure ou à une variable donnée, etc...

En principe l'historique de la session fourni par XCP suffit pour retrouver la valeur d'une variable; mais, par exemple pour réactiver une structure bien précise, il est fastidieux de reconstruire le chemin qui y a mené, surtout si la structure voulue nécessite plusieurs requêtes pour la rendre accessible.

Ainsi, pour accéder à la structure VDEVBLOK de l'unité virtuelle 191 de la machine virtuelle X, nous devons faire successivement :

```
Qualify UTABLE          accès aux machines virtuelles
   =
   =
   =
"Recherche de la machine virtuelle X"
   =
   =
   =
   =
Qualify VCHBLOK  accès aux canaux
   =
   =
   =
   =
"recherche du canal numéro 1"
   =
   =
   =
   =
Qualify VCUBLOK    accès aux unités de contrôle
   =
   =
   =
   =
"Recherche de l'unité de contrôle 9"
   =
   =
   =
Qualify VDEVBLOK         accès aux unités virtuelles
   =
   =
   =
Recherche de l'unité n°7
   =
   =
   =
```

Lorsque la structure VDEVBLOK ne sera plus active, il est assez difficile de retrouver les informations sur l'unité 191 de X. La requête REMEMBER remédie à cet inconvénient en permettant de mémoriser et de nommer un chemin d'accès à une structure c'est-à-dire son contexte . Pour garder en mémoire l'accès à l'unité 191 de X, la requête REMEMBER UNITEX suffit, UNITEX est le nom donné à ce chemin. A tout instant REMEMBER peut être utilisé pour conserver la voie d'accès à une structure, pourvu que cette structure soit active.

g) Activate

La requête ACTIVATE doit être utilisée en conjonction avec REMEMBER. Celle-ci permet de restaurer un chemin (contexte) mémorisé par REMEMBER et de retrouver instantanément la structure "qualifiée" au moment de la requête REMEMBER initiale. Ainsi ACTIVATE UNITEX est équivalent au processus d'accès à l'unité 191 de la machine virtuelle X, décrit au paragraphe f.

Exemple :

```
Remember C2     sauvegarde du contexte actif
Activate C1     activation d'un ancien contexte
```

h) Forget

La requête FORGET est destinée à supprimer certains contextes mémorisés par des requêtes REMEMBER.

i) Search

Lorsque la structure "qualifiée" est un élément d'une liste ou d'un anneau, la requête SEARCH facilite la recherche d'une variable satisfaisant une condition donnée.

Exemple :

Supposons que la structure UTABLE soit qualifiée, et que nous voulions chercher la structure décrivant la marque de l'utilisateur de nom DUPONT.

```
Qualify        UTABLE
Search         USERID  =  "DUPONT"
```

Remarque :

Pour que la recherche soit possible, il est obligatoire que la variable USERID appartienne à la structure UTABLE et qu'elle soit de type CHAR.

j) Find

Cette requête joue le même rôle que SEARCH mais ne concerne pas les variables. Elle permet de retrouver une information dans la mémoire analysée par XCP. Cette requête peut être utile lorsque certaines variables de valeur connue ne possèdent pas de base connue.

Exemple :

Find CORE "PSA"
donne l'adresse mémoire de la chaîne de caractère PSA si elle existe.

Certaines conditions peuvent également être imposées à cette recherche pour qu'elle opère par exemple sur une frontière d'octet, de demi-mot, etc... ou pour fixer les limites de recherche entre 2 adresses données (§ requête SET).

k) Show

Le contenu des variables décrites dans le sous-environnement de définition peut être visualisé au moyen de la requête "SHOW". Celle-ci ne concerne que les variables. Il n'est donc pas possible avec cette requête d'examiner le contenu d'un mot mémoire donné. Le résultat est en conformité avec le type spécifié lors de la définition de la variable. Seules les variables appartenant à la structure active sont accessibles pour cette requête.

```
Qualify UTABLE
Show USERID
USERID   "SØS      "
Show   VGPRS(1)   IØWAIT   RUNUSER   VCHANADD
VGPRS(1)     #0001BA2
IØWAIT       '1'B
RUNUSER      NONEXISTENT
VCHANADD     NONEXISTENT
```

La variable RUNUSER n'est pas accessible car elle est de niveau supérieur. De même VCHANADD est inconnue car elle appartient à la structure VCHBLOK qui est de niveau inférieur.

L'accès aux variables appartenant à une sous-structure se fait de la même façon que dans la plupart des langages de programmation.
Si C appartient à la sous-structure B qui elle-même appartient à la structure principale A, l'accès à C se fait par :

Show B.C

Activation temporaire : Les variables des structures non "qualifiées" peuvent être visualisées en utilisant l'opérateur ->. Nous disons dans ce cas que ces structures sont activées de façon temporaire c'est-à-dire pour la requête SHOW seulement .

Considérons la série des requêtes suivantes :

```
Qualify UTABLE
next 2
Qualify VCHBLOK
Qualify VCUBLOK
Qualify VDEVBLOK
Show  VPNTREAL
```

Pour accéder à la variable VPNTREAL, il a fallu émettre 6 requêtes successives. Ceci est assez fastidieux lorsque l'utilisateur veut s'intéresser principalement à la structure UTABLE.

En utilisant l'activation temporaire, la série des re quêtes ci-dessus se ramène à :

```
Qualify UTABLE
Show  NEXTUSER ->NEXTUSER->VCHSTART->VCULIST->VDEVLIST->VPNTREA
```

La figure 3.4 explique le mécanisme de la requête SHOW avec une activation temporaire.

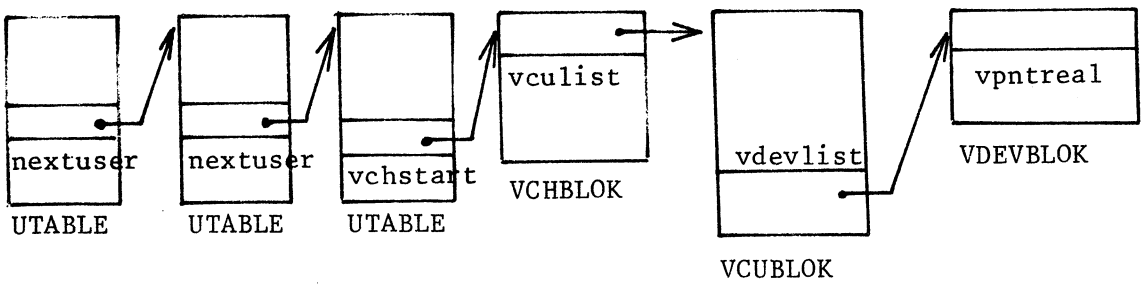


Figure 3.4
Accès à la variable VPNTREAL

Remarque :

Les noms des structures UTABLE, VCHBLOK, VCUBLOK et VPEVBLOK n'apparaissent pas explicitement dans la requête SHOW car ils sont présents dans les déclarations des variables NEXTUSER, VCHSTART VCULIST et VDEVLIST, et par conséquent sont connus par XCP.

l) *Display*

La requête DISPLAY complète la requête précédente car elle permet d'afficher des contenus de mémoire selon des types spécifiés. Ces types sont les mêmes que ceux utilisés dans les définitions des variables sauf pour STRUC et REF.

Exemple :

- Display #160 BYTE ((2)W)
  #160 : #00003600 #00003700

  L'adresse spécifiée peut être un pointeur

- Display #160 → 4 BYTE(H)
  #3700: #ABCD

m) *Dump*

Cette requête permet d'obtenir sur l'imprimante la liste du contenu d'une zone de mémoire sous forme hexadécimale et en caractères EBCDIC équivalent.

n) *Set*

La portée du fonctionnement de XCP peut être modifiée au moyen de la requête SET. Un mot clé suivant le nom de la requête

précise les paramètres que l'on veut modifier.

Exemples :

- set limit 0 4095
  Recherche de la requête "FIND" Se limite à la première page de l'image de CP-67.

- set origin #4000
  L'origine des adresses absolues spécifiées dans les requêtes sera désormais l'adresse #4000.

o) *Query*

La requête QUERY permet de connaître les paramètres de fonctionnnement de XCP. Divers mots-clés aident également à choisir les renseignements voulus.

Exemples :

-Query limit

Cette requête donne les limites de recherche de la requête FIND.

-Query trace

Donne pour toutes les structures "qualifiées" le cheminement utilisé pour accéder à une structure donnée.

En effet, si nous avons utilisé les requêtes ci-dessous :

```
Qualify UTABLE
next 2
Qualify vchblok
Qualify vcublok
next 1
```

La requête QUERY TRACE a comme résultat :

```
UTABLE  WAS  QUALIFIED
#026000   #026B00    #0025400
VCHBLOK   WAS   QUALIFIED
#02A0B0
VCUBLOK   WAS   QUALIFIED
#02B000
```

Les noms et les adresses absolues des structures par lesquelles l'utilisateur "est passé" sont affichés.

p) Call

Cette requête donne la possibilité d'amener en mémoire une copie d'un programme donné depuis un support externe d'informations. Ceci a pour but de permettre à l'utilisateur de XCP d'employer des programmes qui ne sont pas développés pour XCP et de faciliter leur emploi sans modification.

q) Exit

Les caractéristiques des variables sont figées chaque fois que le sous-environnement d'analyse est utilisé. Cette rigidité est dûe à la génération des descripteurs internes, indispensables aux différents mécanismes de XCP. Une modification d'une variable peut entrainer en cascade d'autres modifications. En conséquence, lorsque l'utilisateur veut modifier une variable, il doit revenir dans le sous-environnement de définition au moyen de la requête EXIT.

3.2.1.3.3 Requêtes communes aux sous-environnements

a) Stop

Cette requête permet à n'importe quel moment de quitter XCP.

b) Requests

La liste des requêtes accessibles à l'utilisateur peut être demandé à tout moment par la requête REQUESTS. Elle n'est pas nécessaire lorsque le terminal est un écran de visualisation 2250, car cette liste est affichée en permanence.

c) cp

Cette requête permet à l'utilisateur de faire exécuter les fonctions consoles de CP-67 sans utiliser la touche "attention" du terminal. (§ CP-67/CMS User's Guide).

d) cvt

Cette requête effectue la conversion décimal-hexadécimal ou l'inverse.

e) List

Cette requête déjà décrite plus haut renseigne l'utilisateur sur les définitions de variables.

3.2.2 Extension du langage de base

L'ensemble des requêtes décrites jusqu'alors constitue un noyau minimum pour faciliter le travail de définition et d'analyse, mais est loin d'être complet. En effet, lors de la conception d'un outil de ce type il est difficile de déterminer les besoins exacts des futurs utilisateurs. Aussi doit-on leur donner la possibilité de façonner eux-mêmes les requêtes qu'ils désirent.

Par le mécanisme des macro-requêtes, nous pensons donc atteindre 2 objectifs :

- enrichir à volonté l'ensemble des requêtes de base
- adapter au mieux l'outil aux besoins.

Dans ce qui suit, nous donnons les caractéristiques essentielles du langage d'extension.

3.2.2.1 Forme des macro-requêtes

Une macro-requête est un ensemble de requêtes du langage de base ou du langage d'extension; ces dernières ont pour rôle essentiel d'enchainer l'exécution des requêtes de base, de façon conditionnelle ou inconditionnelle et de contrôler le résultat de chaque exécution d'une requête de base.

Le langage d'extension permet également d'effectuer des opérations sur des variables arithmétiques et des variables caractères.

Pour XCP, une macro-requête se trouve dans un fichier dont le nom est préfixé par le caractère ε et dont le type est MAC. Elle comprend des requêtes du langage de base et du langage d'extension mais peut également contenir des appels à des macro-requêtes. Ceci autorise une récursivité totale dont le niveau maximal n'est limité que par l'espace mémoire disponible au moment de l'utilisation.

Exemple de macro-requête

La requête LIST permet d'obtenir le contenu d'une macro-requête

```
list  εetat  mac      εetat et le nom de la macro-requête
εSET  MPRØC  OFF      requête du langage d'extension
NEXT 1                } requêtes de base
SHOW  VMSTATUS        }
```

3.2.2.2 Eléments du langage d'extension

Le langage d'extension se compose de requêtes, de variables de travail de types arithmétique et caractère et de variables de liaison. La forme des requêtes est identique à celle des requêtes du langage de base mis à part le nom qui est préfixé par le caractère ε.

3.2.2.2.1 Les variables globales et locales

a) Variables arithmétiques

A chaque niveau de récursivité sont prédéfinies 10 variables εi0,...εi9 de type entier, avec lesquels l'utilisateur peut effectuer des calculs au moyen des expressions arithmétiques.

Exemple :

```
εi1 = (εi2+24)/εi1
NEXT  εi1
-
-
```

Ces variables ont une portée limitée à un niveau de récursivité et sont toujours initialisées à zéro pour chaque niveau.

De façon similaire sont définies 10 variables globales εG0,.... εG9 type entier, mais dont les valeurs sont accessibles à travers tous les niveaux. Elles ont pour valeur 0 à l'initialisation de XCP et ne changent pas de valeur d'un niveau à un autre sans modification explicite.

b) Variables caractères

L'utilisateur peut définir des variables de type caractère

dont la valeur maximale est de 8 caractères. La déclaration de telles variables est implicite et se fait à la première affectation; sa portée est celle de la macro-requête dans laquelle elle est définie. Enfin la seule opération possible sur ces variables est l'opération de concaténation représentée par l'opérateur |.

Exemple :

```
εA = "XYZ"        déclaration implicite de εA

εB = εA|"X"       déclaration de εB et affectation de la
                  valeur "XYZX"
```

3.2.2.2.2 Les variables de liaison

Contrairement aux variables de travail, arithmétiques ou caractères, les variables de liaison ne peuvent être utilisées en partie gauche d'une affectation, car à tout instant, elles portent une valeur déterminée.

Ces valeurs sont fonction des paramètres d'appels de la macro-requête et de l'exécution des requêtes qui la composent. Elles servent à paramétrer l'utilisation des macro-requêtes ou à établir des liens entre les requêtes de base et celles du langage d'extension.

a) Valeurs des paramètres d'appel

Une macro-requête peut être utilisée en donnant son nom suivi de paramètres d'appel.

Exemple :

```
εM    "XYZ"   1   ABC
```

Une requête du langage d'extension peut se référer à ces paramètres de position en utilisant les variables ε0 ε1..... ε30 , qui désignent

respectivement le nom de la macro-requête, le 1er paramètre et.... jusqu'au trentième.

Remarques :

- Un paramètre d'appel peut être omis, mais doit être signalé absent en mettant à l'appel de la macro-requête, le caractère ' à la place du paramètre.

  εM "XYZ" ' ABC

  Dans l'exemple ci-dessus le deuxième paramètre de la macro εM est omis, c'est-à-dire que dans toute la définition de la macro-requête, la chaîne vide devient la valeur de ce paramètre.

- Les valeurs des paramètres sont redéfinies chaque fois que εPARM et εREAD sont utilisées. (§ requêtes εPARM et εREAD).

b) *Nombres de paramètres d'appel*

La variable εPARMNUM a pour valeur le nombre de paramètres d'appel d'une macro-requête. Sa valeur est accessible à tous les niveaux de récursion et elle n'est modifiée que par εPARM et εREAD. (§ requêtes εPARM et εREAD).

c) *Niveau de récursivité*

La variable εLEVEL indique le niveau de récursivité de la macro-requête en cours d'exécution. Le niveau initial est le niveau 1.

d) *Numéro de ligne*

La variable εLINENUM contient le numéro de l'élément plus un

de la macro-requête en cours d'exécution.

e) Code d'erreur d'exécution fourni par les requêtes de base

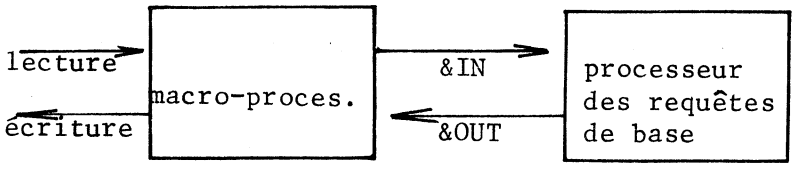
Chaque exécution d'une requête de base se termine en mettant dans le registre général 15 une valeur numérique indiquant comment s'est effectuée l'exécution et cette valeur est affectée à la variable εRETCODE. Ainsi dans une macro-requête, l'utilisateur peut connaître le résultat de l'exécution d'une requête en contrôlant la valeur de εRETCODE.

3.2.2.2.3 Variables de contrôle

Avec l'organisation choisie pour réaliser l'environnement XCP (§ Chapitre 4), il n'est pas possible dans le langage d'extension de se référer aux résultats d'exécution requêtes de base. Pour permettre un lien entre les processeurs, 2 variables εIN et εOUT sont définies. Grâce à ces deux variables globales (de type caractère) l'utilisateur peut contrôler ou surveiller l'exécution des requêtes de base.

εIN contient la ligne d'entrée et au cours de l'exécution d'une macro-requête, une lecture à la console est effectuée.

εOUT contient la ligne de sortie lors d'une écriture effectuée par une requête de base contenue dans une macro-requête.



D'après le schéma ci-dessus, les variables εIN et εOUT rendent accessibles les informations manipulées par les requêtes de base. L'accès aux sous-chaînes de εIN et εOUT est également possible, en spécifiant leur position dans la chaîne. La longueur maximale d'une sous-chaîne est fixée à 8 caractères.

Exemple :

```
Show  USERID
USERID  "OPERATOR"
εNOM = εOUT(9)
```

La requête SHOW demande l'impression de la valeur de la variable USERID. La deuxième ligne de l'exemple représente la sortie effectuée par la requête SHOW. La valeur de εOUT à cet instant est le contenu de toute cette 2ème ligne.

Enfin à la variable εNOM est affectée la sous-chaîne εOUT(9) dont la valeur est OPERATOR.

3.2.2.2.4 Les différentes requêtes

Ces requêtes du langage d'extension sont classées en différents groupes selon leur fonction.

a) Affectation des paramètres de fonctionnement

La requête εSET permet de modifier les valeurs données par défaut aux paramètres de fonctionnement du macro-processeur; l'utilisateur a donc la possibilité d'imposer de nouvelles valeurs à ERROR, MPROC, REQ, PRT, et PROC qui sont les différents paramètres de εSET. Les valeurs possibles de ces paramètres sont ON ou OFF.

ERROR ON : le macro-processeur doit visualiser la valeur de εRETCODE lorsque celle-ci est différente de zéro (erreur dans l'exécution d'une requête de base).

OFF : pas de message d'erreur.

MPROC ON : le macro-processeur doit visualiser la requête du langage d'extension chaque fois qu'il l'exécute. Ceci correspond à demander un historique des requêtes exécutées.

OFF : pas d'impression des requêtes

REQ ON : même effet que MPROC ON mais concerne les requêtes de base.

OFF : pas d'impression des requêtes de base.

PRT ON : le macro-processeur doit écrire sur l'imprimante un double des écritures effectuées sur la console.

OFF : pas de sortie sur l'imprimante.

PROC ON : les écritures provenant d'une requête de base doivent être effectuées sur le terminal.

OFF : Toutes les écritures sur le terminal sont supprimées.

Remarques

- tous les paramètres de εSET sont indépendants
- le contenu d'une ligne destinée à l'impression peut toujours être connue à l'aide de εOUT même si la requête εSET PROC OFF a été émise.

b) Transfert de contrôle

3 requêtes permettent d'effectuer des ruptures de séquences à l'intérieur d'une macro-requête ou de commander un retour à une macro-requête d'un niveau inférieur. Ce sont εSKIP εGØTØ εRETURN.

εSKIP n — supprime l'exécution des n requêtes qui suivent εSKIP. Ces requêtes sont soit des requêtes de base soit des requêtes du langage d'extension.

Exemple :

εSKIP 3
ignore les 3 requêtes qui suivent εSKIP.

εGOTØ Etiquette
commande une rupture de séquence vers l'étiquette donnée ou à un numéro de requête donné.

Exemple :

- εGØTØ LABEL
  LABEL : ...
  } Rupture de séquence jusqu'à la requête portant l'étiquette LABEL

- εGØTØ 3 — exécuter la requête numéro 3

- εGØTØ εTOP — exécuter la 1ère requête (εTOP est un mot-clé)

Remarque

Les étiquettes sont une suite de 8 caractères alphanumériques. Elles doivent être suivies du signe : lorsqu'elles référencent une requête.

εRETURN n commande le retour au niveau de récursivité inférieur avec le code d'erreur n.

Exemple :

εRETURN 5

retour à la macro-requête appelante avec un code d'erreur égal à 5.

e) Requête itérative

La requête εLOOP permet de renouveler l'exécution d'une séquence d'instructions un certain nombre de fois ou jusqu'à ce que la condition spécifiée soit satisfaite . A l'intérieur de la portée εLOOP, d'autres requêtes εLOOP peuvent être utilisées jusqu'à un niveau d'imbrication de 4.

Exemples :

```
εLOOP  3  2
-
-
-
```

Exécute les 3 requêtes suivantes 2 fois

```
εLOOP  FIN  εI1 > 4
-
-
-
FIN :
```

Exécute les requêtes jusqu'à l'étiquette FIN jusqu'à ce que la variable εI1 ait une valeur égale à 5.

d) Requêtes conditionnelles

Les requêtes εIF et εON permettent de contrôler les valeurs des variables numériques et caractères et d'exécuter d'autres requêtes si les tests effectués sont concluants. Le fonctionnement de εIF et εON ressemble de très près à celui des instructions IF et ON des langages évolués.

εIF condition action

Chaque fois que εIF est exécutée, la condition spécifiée est vérifiée si elle est satisfaite, alors l'action est exécutée. Cette action est réalisée par une requête de base ou par une requête du langage d'extension.

εON condition action

Après l'exécution d'une requête εON, la condition spécifiée est vérifiée au cours de l'exécution des requêtes de base ou du langage d'extension. Si la condition est satisfaite, alors l'exécution de l'action spécifiée aura lieu. Dans ce cas, l'exécution des requêtes est suspendue et sera reprise lorsque l'action est terminée.

Exemples :

εIF εG1 = 3 ACTIVATE P1

Si la variable de base ACTIVATE est exécutée.

εON εOUT(1)="RIEN" εEXIT

Si au cours de l'exécution des requêtes, la sortie du processeur est égale à la chaîne "RIEN" alors l'exécution de la macro-requête se termine (εEXIT).

e) Requête ineffective

La requête εCONTINUE est utilisée comme requête de non-opération en conjonction avec les requêtes εGOTO, εLOOP et εON .

Exemples :

```
εLOOP   LABEL   εI1 > 3
  -
  -
LABEL : εCONTINUE
```

εCONTINUE est utilisée ici pour repérer une séquence pour εLOOP

εON εERROR εCONTINUE

Si une erreur est trouvée dans l'exécution d'une requête de base alors l'action à prendre est une non-opération, ceci revient à ignorer l'erreur et à continuer l'exécution.

f) Données en entrée

Diverses requêtes permettent de redéfinir les paramètres d'appel d'une macro-requête, de lire des requêtes à partir du terminal ou de stocker à l'avance des lignes à imprimer.

εPARM les valeurs des paramètres d'appel ε1 ε2.... ε30 d'une macro-requête peuvent être redéfinies au cours de l'exécution de la macro-requête par εPARM.

εREAD permet de spécifier que les n requêtes suivant la requête εREAD doivent être lues à partir du terminal. Si le paramètre de εREAD est PARM, alors une seule ligne est lue et les paramètres ε1... ε30 sont redéfinis à partir des valeurs contenues dans cette ligne.
Les requêtes et les valeurs lues par εREAD sont traitées par le macro-processeur de la même façon que les autres requêtes du fichier de type MAC.

εSTACK {FIFO / LIFO} cette requête est utilisée pour pré-enregistrer une ligne. Cette ligne peut être ensuite lue soit par la requête εREAD soit par une requête de base, sans que l'utilisateur ait à taper la ligne. Celle-ci est traitée comme une requête de la macro-requête.
L'utilisateur peut également spécifier l'ordre de rangement de ces requêtes: FIFO première requête enregistrée, première requête lue ou LIFO dernière requête enregistrée, dernière requête lue.

εBEGSTACK cette requête permet de pré-enregistrer les requêtes qui auraient pû être obtenues par un ordre de lecture au terminal à l'aide d'une requête de base. L'utilisateur peut, comme avec εSTACK, spécifier l'ordre de lecture de ces requêtes. Celles-ci ne sont pas analysées par le macro-processeur pour les substitutions des paramètres et de ce fait sont exécutées telles qu'elles sont. Elles ne doivent pas commencer par le mot clé εEND qui indique la fin des requêtes à enregistrer.

Exemples :

- εPARM 3 ABC
  les paramètres d'appel ε1 et ε2 ont pour valeur 3 et ABC

- Considérons la macro-requête εTEST suivante :

```
εPRINT  ε1
εI1 = 5
εREAD  1
εPRINT FIN
```

L'exécution de cette macro-requête a pour effet :

```
εtest   a .............  appel de la macro
A .....................  exécution de la 1ère requête
.......................  exécution de la 2e requête
.......................  exécution de la 3e requête qui
                          effectue une lecture au terminal.
εprint  εI1  εI2 ......  requête lue au terminal
5 0 ...................  Effet de l'exécution de la requête
                          lue par εREAD
FIN ...................  exécution de la dernière requête.
```

- εBEGSTACK

```
A BIT(3)
B REF(C)
END
```

εENDSTACK
DEFINE S STRUC BASE(B1)

Lorsque la requête DEFINE est exécutée, l'utilisateur n'a pas à taper la suite de la définition de la structure S, puisqu'elle est pré-enregistrée par εBEGSTACK.

g) Impression sur le terminal

Deux requêtes permettent d'effectuer et de contrôler la sortie sur le terminal. Ce sont εPRINT et εSPACE.

εPRINT imprime sur le terminal une ligne dont le contenu est spécifié en paramètre. Avant de commander l'opération d'impression, le macro-processeur effectue si besoin est une substitution des variables qui s'y trouvent.

εSPACE n imprime n lignes blanches sur le terminal pour donner une présentation plus claire des résultats obtenus.

3.2.2.2.5 Utilisation en machine de bureau

L'utilisateur peut se servir du terminal connu d'une machine de bureau pour effectuer des calculs sur les variables globales εG0...εG9 Ces calculs sont présentés sous forme d'expressions arithmétiques qui sont immédiatement analysées et évaluées selon la structure parenthésée et la priorité des opérateurs présents. Le résultat de l'expression n'est donné que sur demande explicite.

Exemple :

- εg1 = 256/3 affectation de εG1
- εg0 = (εg1 + εg2 * #44)/(εg3 + 7) affectation de εG0
- εg0 demande d'impression de la valeur de εG0
  256 (#100)

3.3 UTILISATION DU TERMINAL GRAPHIQUE

XCP peut être utilisé indifféremment avec un terminal du type machine à écrire ou encore avec un écran de visualisation IBM 2250. Le choix se fait à l'appel de XCP en spécifiant le type du terminal.

Les possibilités obtenues avec le 2250 sont évidemment plus importantes qu'avec un terminal classique, puisque nous pouvons disposer de l'écran,des touches de fonction et du pointeur optique. Avant d'examiner leurs rôles respectifs, nous étudions le découpage logique de l'écran.

3.3.1 Les différentes zones de l'écran

Su la figure 3.5 nous distinguons 3 zones matérialisées par 2 traits horizontaux.

1/ Zone des requêtes disponibles

A chaque instant, XCP affiche dans la zone du haut de l'écran la liste des requêtes disponibles pour l'utilisateur à cet instant. Celle-ci est modifiée automatiquement chaque fois que l'on rentre dans un sous-environnement donné.

2/ Zone des renseignements généraux

Cette zone comprend 3 lignes qui contiennent dans l'ordre :

- la dernière requête ou macro-requête exécutée

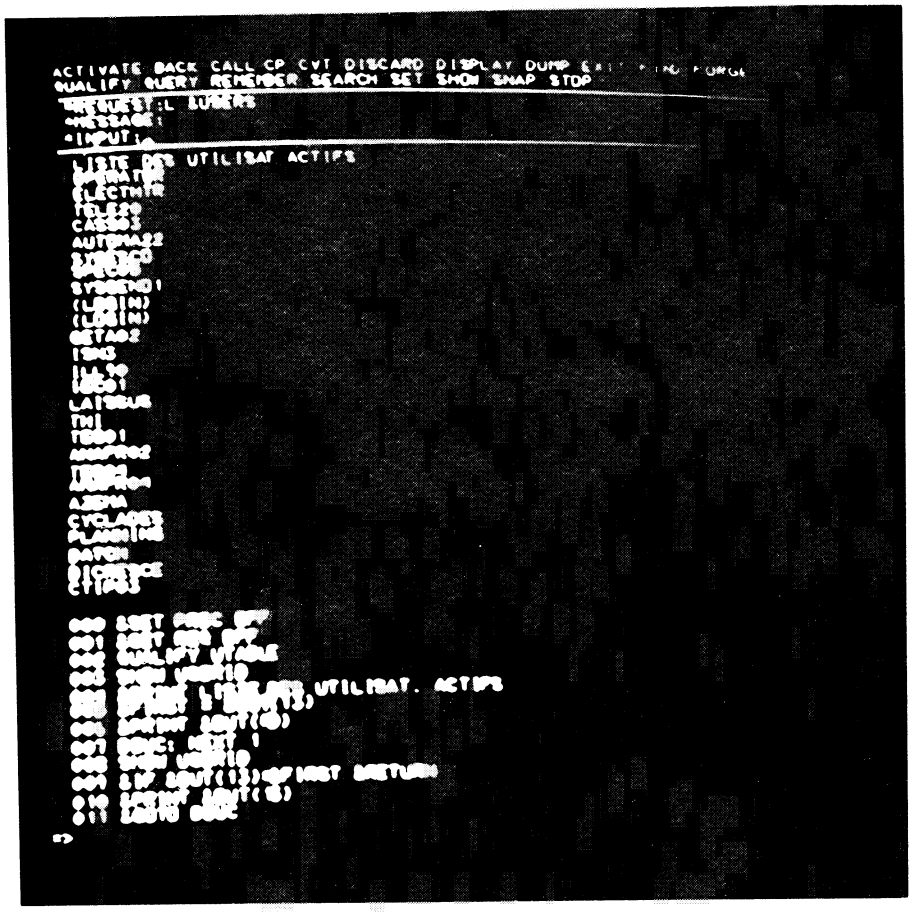


Figure 3.5

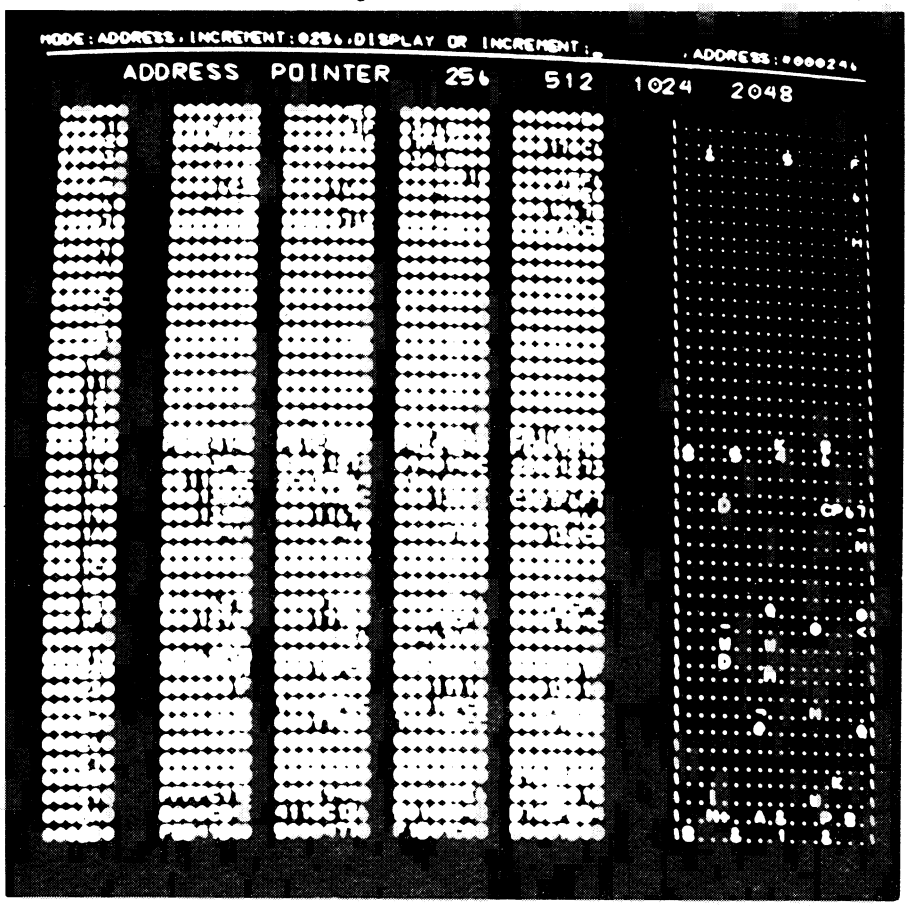


Figure 3.6

- le message d'erreur éventuel qui l'accompagne,

- la ligne sur laquelle apparait la prochaine requête entrée, avec le curseur permettant de repérer le dernier caractère tapé.

3/ Zone d'affichage des résultats

Cette zone comprend 42 lignes et sert à l'affichage des résultats fournis par une ou plusieurs requêtes ou macro-requêtes. La dernière ligne affichée est repérée à gauche par les caractères =>. (§ modes PAGE et ROLL).

3.3.2 Touches des fonctions programmées

Les touches des fonctions programmées n'ont pas d'équivalent sur un terminal classique; elle facilitent l'usage du 2250 et rendent plus aisé l'emploi de XCP. Parmi l'ensemble des touches disponibles, seules les touches allumées ont une signification. L'éclairage de ce sous-ensemble est lié au sous-environnement utilisé et est fonction des conditions de fonctionnement.

Nous donnons dans ce qui suit, leur numéro d'ordre, leur nom symbolique, et leur signification lorsqu'elles sont utilisées :

Touche 0 CLEAR — Provoque la remise à blanc de la zone d'affichage des résultats. Cette touche est toujours allumée.

Touche 1 COPY — Reproduit sur l'imprimante virtuelle l'image exacte de l'écran. Cette touche est toujours allumée.

Touche 2 ROLL

Force l'utilisation de l'écran en mode ROLL. Cette touche est allumée si l'écran fonctionne en mode PAGE et est éteinte dans le mode ROLL.

Mode ROLL : lorsque la zone d'affichage est saturée, elle est décalée de moitié vers le haut. Les lignes libérées sont remises à blanc en vue d'un affichage ultérieur.

Touche 3 PAGE

Force l'utilisation de l'écran en mode PAGE. Cette touche est allumée lorsque l'écran est en mode ROLL.

Mode PAGE : lorsque la dernière ligne de la zone d'affichage est écrite, les lignes suivantes sont alors écrites en haut de la zone et surchargent les lignes existantes. (Recouvrement circulaire).

Touche 4 CLOSE

Provoque l'impression effective des copies obtenues précédemment par la touche COPY. Cette touche efface toute trace des COPY précédentes, et reste toujours allumée.

Touche 10 EXAMON

Provoque l'entrée dans le sous-environnement graphique d'analyse (§ 3.3.3) Elle est éteinte lorsque le sous-environnement est activé.

Touche 11 EXAMOFF

Correspond à une requête du sous-environnement graphique d'analyse. Elle n'est allumée que dans ce sous-environnement (§ 3.3.3).

Touche 12 DOWN — Mêmes remarques que pour EXAMOFF .

Touche 13 UP — Mêmes remarques que pour EXAMOFF.

Touche 14 DISPLAY — Mêmes remarques que pour EXAMOFF

Touche 15 INCREMENT — Mêmes remarques que pour EXAMOFF

Touche 22 SYNONYM — Cette touche,toujours allumée,permet de rendre équivalente une des touches 24 à 31 à une requête qui vient d'être entrée par le clavier alphanumérique.

Pour ce faire, l'utilisateur doit procéder de la façon suivante :

1) Taper une requête au clavier alphanumérique
2) Appuyer sur la touche SYNONYM.
3) Appuyer sur une des touches éteintes 24 à 31. La touche choisie est alors allumée et a la même fonction que la requête d'entrée.

Lorsqu'une des touches 24 à 31 préalablement rendue équivalente à une requête est utilisée, elle provoque l'exécution de la requête équivalente.

Remarque

Pour libérer une touche rendue équivalente, il suffit de procéder comme ci-dessus mais avec comme requête d'entrée une requête nulle.

Touche 23 SYNLIST — Fait apparaître dans la zone d'affichage la liste des touches 24 à 31 et leur requête équivalente. Cette touche est toujours allumée.

Touche 24 à 31 — Utilisées en conjonction avec la touche SYNONYM.

3.3.3 LE POINTEUR OPTIQUE

Le pointeur optique sert comme moyen de désignation des objets affichés sur l'écran. Pour des raisons de compatibilité d'utilisation de XCP avec les différents terminaux, le pointeur optique n'est utilisé que dans le sous-environnement graphique d'analyse.

3.3.4 LE SOUS-ENVIRONNEMENT GRAPHIQUE D'ANALYSE

Ce sous-environnement est activé par la touche EXAMON et par conséquent n'est accessible que si le terminal de dialogue est un IBM 2250. Il permet l'utilisation du terminal graphique comme moyen rapide pour examiner une image mémoire ou un instantané de CP-67. Cet examen se fait uniquement avec les touches des fonctions programmées (11 à 15) et le pointeur optique. Une nouvelle image est affichée lorsque l'on rentre dans ce sous-environnement. L'ancienne image est conservée et sera restaurée lorsque la touche EXAMOFF est appuyée.

Dans ce sous-environnement l'écran peut être comparé à une "fenêtre" sur une liste d'un dump classique (fig.3.6). Le déplacement de cette fenêtre se fait, soit par les touches UP, DOWN et DISPLAY, soit par le pointeur optique.

1/ Présentation de l'écran

L'écran est partagé en 2 parties :

- La partie supérieure (1 ligne) est utilisée pour afficher les mots clés que l'utilisateur peut désigner grâce au pointeur optique; ce sont :

  ADDRESS POINTER 256 512 1024 2048

- La partie inférieure (42 lignes) est utilisée pour afficher la mémoire examinée sous forme hexadécimale et caractères EBCDIC équivalents.

2/ Touches de fonctions

Seules les touches 11 à 15 sont allumées dans ce sous-environnement.

Touche 11 EXAMOFF — Permet de quitter le sous-environnement graphique d'analyse. L'ancienne image est restaurée et les touches autres que 11 à 15 sont de nouveau rendues actives.

Touche 12 DOWN — Provoque l'affichage de la zone de mémoire dont l'adresse est l'adresse de la première ligne affichée augmentée de l'incrément en cours.

Touche 13 UP — Est le similaire de la touche DOWN mais permet d'afficher une zone de mémoire dont l'adresse est égale à l'adresse de la première ligne affichée diminuée de l'incrément en cours.

Touche 14 DISPLAY Provoque la lecture d'une adresse au clavier alphanumérique du 2250 et l'affichage de la zone de mémoire correspondante à cette dernière.

Touche 15 INCREMENT Provoque la lecture au clavier alphanumérique du 2250 de la valeur de l'incrément.

3/ Le pointeur optique

Le pointeur optique sert à désigner :

- des mots-clés qui représentent les paramètres de fonctionnement du sous-environnement ou les modes de fonctionnement du pointeur optique lui-même.

- le contenu d'une mémoire affichée sur l'écran.

a) Paramètres de fonctionnement

Les 4 nombres (256 512 1024 2048) affichés en haut de l'écran définissent la valeur de l'incrément pour calculer l'adresse de la prochaine zone de mémoire à afficher (§ touches UP et DOWN).

b) Modes du pointeur optique

Le mode de fonctionnement du pointeur optique est choisi en désignant les mots clés ADDRESS et POINTER avec le pointeur optique lui-même.

Mode ADDRESS : lorsqu'un chiffre hexadécimal ou un caractère EBCDIC est désigné, l'adresse correspondante est affichée et le curseur est inséré sous l'objet désigné pour repérer son remplacement.

Mode POINTER : dans ce mode le pointeur optique doit désigner un objet qui devient l'adresse de la prochaine zone à afficher. Ceci est très pratique pour suivre le cheminement dans une liste.

C H A P I T R E 4

PRINCIPES DE FONCTIONNEMENT DE XCP

C H A P I T R E 4

Dans le chapitre précédent nous avons expliqué comment un utilisateur voit l'environnement XCP par son langage des requêtes et ses mécanismes permettant une exploration des structures des données du système CP-67. Dans ce chapitre, nous explicitons la conception de XCP lui-même, c'est-à-dire la structure interne que nous avons choisi.

4.1 ORGANISATION GENERALE

L'environnement XCP se décompose schématiquement (fig.4.1) en 3 parties distinctes en fonction de nos remarques du chapitre 2 (voir paragraphe 2.3).

1/ Un processeur destiné à gérer l'unité d'entrée/sortie

servant de terminal de dialogue. Ceci est dû au fait que XCP doit pouvoir fonctionner indifféremment avec un terminal de type machine à écrire ou avec un terminal graphique IBM 2250.

2/ Un processeur pour les requêtes de base.

Il est le noyau même de XCP, puisqu'il remplit les fonctions de description des variables et d'analyse des images mémoires ou des instantanés Il est indépendant des deux autres processeurs composant l'environnement XCP.

3/ Un macro-processeur pour le langage d'extension

dont le rôle est de décoder et d'exécuter les macro-requêtes. De par sa conception, il contrôle également la marche du processeur des requêtes qui l'ignore complètement.

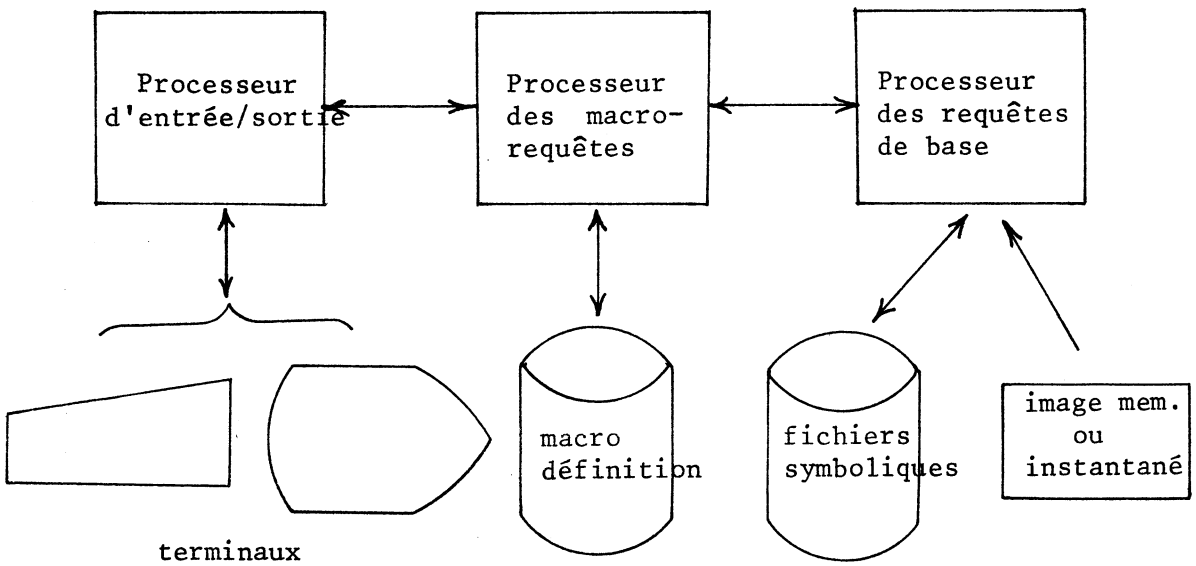


Figure 4.1

Les processeurs de X.C.P

4.1.1 Les différents modules de XCP

Les différents modules de XCP sont programmés en langage d'assemblage 360 [Ref.21] et en PL 360 [Ref.37]. Pour des raisons évidentes de mise au point et de maintenance, nous aurions voulu écrire notre

système dans un langage évolué. Malheureusement au démarrage de notre projet, il n'existait aucun langage qui répondait aux spécifications désirées. Nous avons donc fait un compromis en utilisant le langage d'assemblage 360 et PL360. Malgrè sa simplicité et sa clarté ce dernier comporte quelques lacunes: impossibilité de définir des macro-instructions, impossibilité d'inclure des définitions de données à partir d'une description déjà existante (pas d'équivalent de la pseudo-instruction COPY de l'assembleur). Dans la mesure du possible les programmes (1/3 environ) sont écrits en PL360, chaque fois que les difficultés inhérentes au langage peuvent être levées.

Du côté du système CMS, sous lequel notre environnement fonctionne, nous avons utilisé largement ses possibilités pour la réalisation des différents modules.

4.1.1.1 CONSOL: processeur des entrées/sorties avec le terminal

Toute demande d'entrée-sortie destinée au terminal est prise en charge par ce module qui a pour but de gérer le trafic des messages entre les processeurs et le terminal. CONSOL est indépendant des autres processeurs et peut donc fonctionner avec n'importe quel autre environnement conversationnel. Pour ce faire, CONSOL fait appel aux fonctions du système CMS pour réaliser les entrées/sorties classiques (2741) et au module IO2250 pour la partie gestion des interruptions et des entrées/sorties de l'unité d'affichage graphique 2250.

La partie traitement du crayon optique des touches de fonctions programmées de l'unité 2250 sera détaillée au paragraphe 4.1.4.

4.1.1.2 MAIN et DECODE : processeur des requêtes de base

La partie responsable du traitement des requêtes de base est l'ensemble de modules MAIN, DECODE et $R_1 \ldots R_n$ fonctionnant suivant le schéma ci-dessus (fig.4.3).

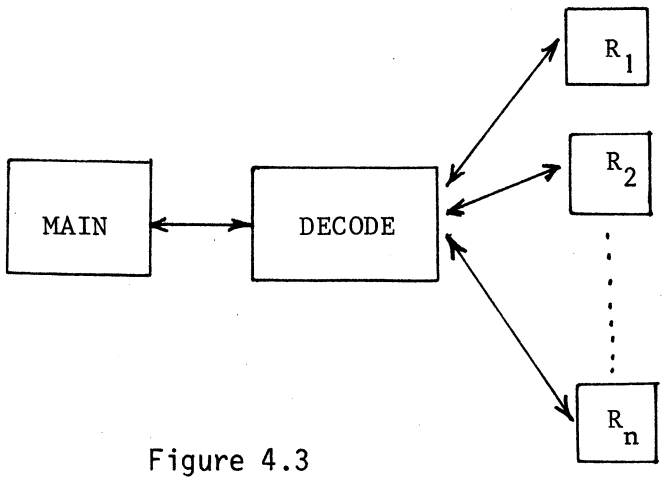


Figure 4.3

Organisation du processeur des requêtes de base

Le module MAIN réalise :

- l'initialisation de l'environnement, c'est-à-dire calcul de la taille de la mémoire libre disponible, lecture des fichiers symboliques,etc...

- la génération des descripteurs internes des variables symboliques et la construction des liens entre ces descripteurs.

- la lecture des requêtes.

Lorsqu'une requête est émise par l'utilisateur, le contrôle est donné au module DECODE. Celui-ci analyse la requête d'entrée et détecte les erreurs syntaxiques les plus grossières, il donne ensuite le contrôle au module qui a pour nom le nom de la requête.

Les différents modules R1...RN ont pour rôle unique d'exécuter le traitement correspondant à la requête. Ils sont indépendants les uns des

autres, et par conséquent peuvent être modifiés, supprimés ou créés aisément. Le seul lien existant entre les 3 parties est une zone de données commune contenant tous les enseignements nécessaires au bon fonctionnement des différents modules.

4.1.1.3 MACPROC processeur des macro-requêtes

D'après la figure 4.1 en série avec le processeur principal, nous trouvons le macro-processeur. Ce schéma de fonctionnement du processeur et du macro-processeur est à l'opposé de celui adopté dans certains systèmes (fig.4.4).

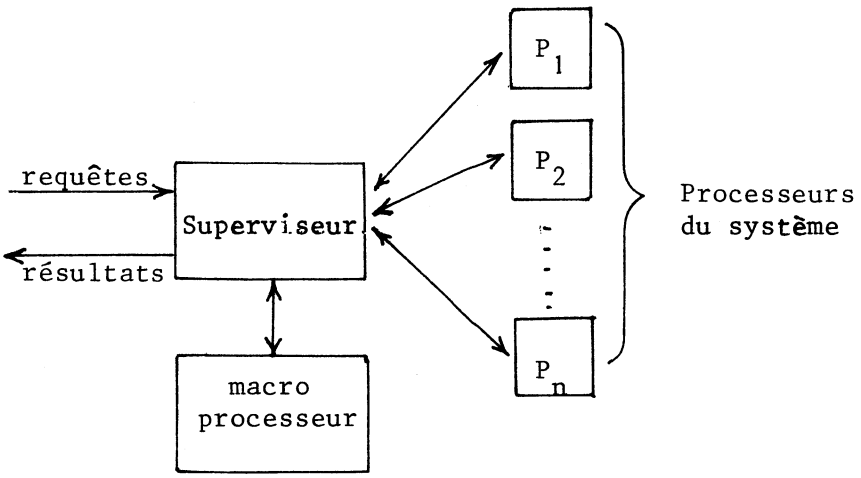


Figure 4.4

Relation Superviseur - Macro-processeur

En effet, dans ces systèmes, le macro-processeur est au même niveau que le décodeur des requêtes du superviseur et en parallèle avec lui.

Pour éviter les inconvénients dûs à cette dépendance, nous avons adopté dans XCP l'organisation suivante pour le macro-processeur :

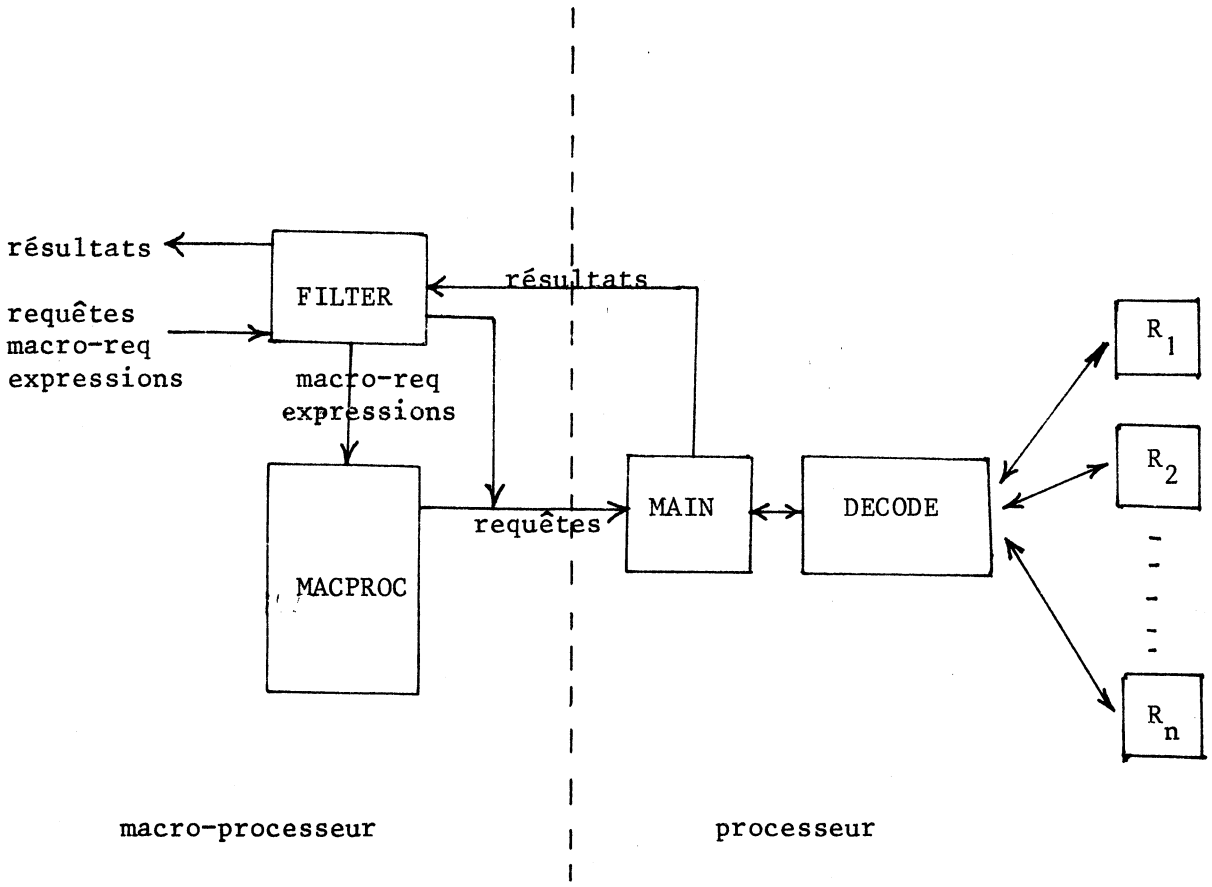


Figure 4.5

Relations entre Macro-processeur
et Processeur

- FILTER est le module qui joue le rôle de filtre et de ce fait "surveille" les entrées/sorties réalisées par les processeurs.

- MACPROC est le module de décodage et d'exécution des macro-requêtes.

La seule contrainte dûe à l'organisation choisie est que toutes les demandes d'entrée/sortie doivent passer obligatoirement par le filtre.

Lorsqu'une requête est reçue depuis le terminal, elle est examinée par FILTER et aiguillée vers les modules responsables. Cette requête peut être :

- une requête de base; dans ce cas elle est transmise directement au processeur principal pour être analysée et exécutée.

- une macro-requête; dans ce cas elle est dirigée vers MACPROC qui vérifie son existence effective. Si un fichier de même nom et de type MAC existe, alors son contenu est :

  . soit analysé et exécuté par MACPROC (requêtes du langage d'extension
  . soit transmis au processeur principal.

- une expression arithmétique (affectation d'une variable globale) ou un nom d'une variable globale (demande d'impression de la valeur). L'environnement est alors utilisé comme machine de bureau et la requête est traitée par MACPROC

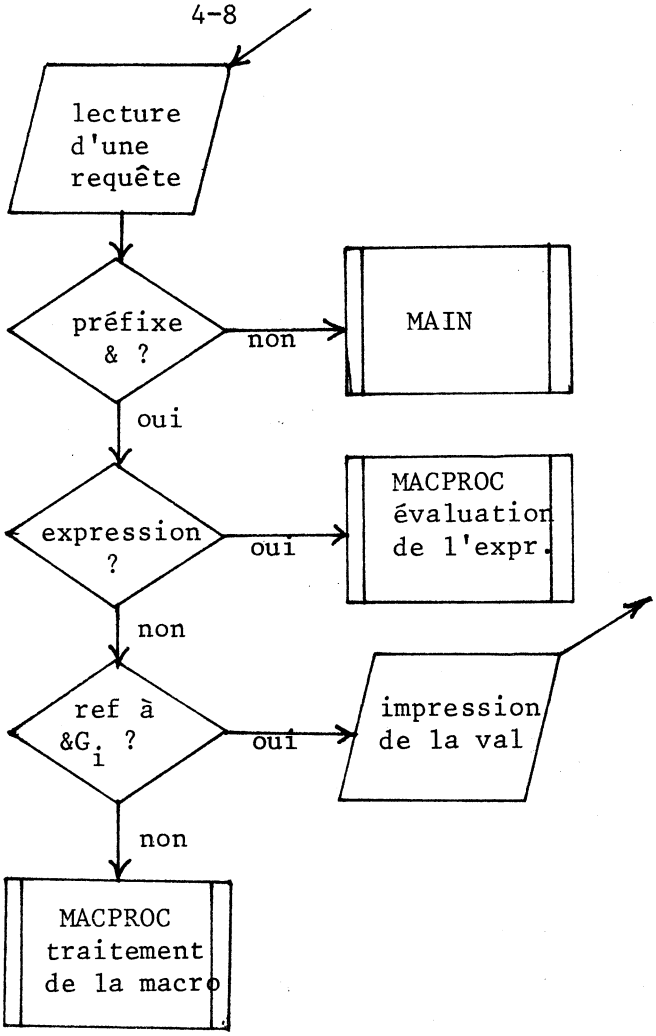


Figure 4.6

Reconnaissance et traitement d'une ligne

D'après le principe de fonctionnement adopté, nous voyons qu'il y a indépendance entre les 2 parties processeur et macro-processeur. De ce fait, le macro-processeur peut être utilisé avec n'importe quel environnement conversationnel. La seule contrainte imposée sur le processeur est que les entrées/sorties doivent être faites par des macro-instructions particulières forçant un appel au module FILTER.

4.1.2 Définition de la machine virtuelle nécessaire au fonctionnement de XCP.

L'environnement XCP est destiné à fonctionner sur une machine virtuelle générée par CP-67; sa configuration se présente ainsi :

- une mémoire centrale de 256K octets plus la taille mémoire de la machine réelle. Dans notre cas nous avons 1280K octets, car le 360/67 du CICG dispose de 1024K octets.

- une unité virtuelle de "spooling" appelée REMOTE PRINTER-READER destinée à lire le fichier de "spooling" qui contient l'image mémoire à analyser.

- une unité d'affichage graphique IBM 2250. Cette unité est facultative et peut toujours s'ajouter dynamiquement à la configuration de la machine virtuelle par la commande ATTACH [Ref.14] de l'opérateur du système. De ce fait elle n'a pas besoin d'être définie explicitement dans la configuration par défaut de la machine virtuelle.

- une unité de bande magnétique est nécessaire lorsque les images mémoire ou les instantanés sont stockés sur bande. Comme le terminal graphique, cette unité se définit dynamiquement par la commande ATTACH.

Quant au système CP-67, nous avons introduit un module supplémentaire destiné à exécuter les requêtes SNAP émises par XCP. Les détails de l'implantation de ce mécanisme sont donnés au paragraphe 4.3.

4.1.3 Génération de XCP et conventions de liaison entre les différents modules.

Tous les paramètres relatifs à la génération d'une version

de XCP sont contenus dans un fichier et sont définis au moyen des variables d'assemblage [Ref.21]. Nous y trouvons entre autres la taille mémoire du 360-67, l'adresse des différentes unités d'entrée/sortie, etc... Cette paramétrisation ne concerne que les programmes écrits en langage d'assemblage. De la même façon, les définitions des données communes aux processeurs sont contenues dans un fichier et sont accessibles

- statiquement, lors de la création des modules, par la pseudo-instruction COPY de l'assembleur

- et dynamiquement lors de l'exécution des modules.

Les liens entre les modules respectent les conventions du système OS/360 [Ref.22] c'est-à-dire que le module appelant doit fournir une zone de sauvegarde dans laquelle le module appelé sauve les registres généraux.

Les appels entre modules se font par la macro-instruction CALL d'OS/360, tandis que les entrées/sorties se font par les macro-instructions que nous avons définies CSLIN et CSLOUT.

4.1.4 Gestion des ressources disponibles

La plupart des ressources dont dispose XCP sont gérées par le système CMS. Pour diverses raisons que nous expliciterons, nous avons introduit un deuxième niveau de gestion des 2 ressources, mémoire libre et unité 2250.

4.1.4.1 Gestion de la mémoire libre

Le module STORAGE de CMS s'avérant mal adapté à nos besoins, nous avons réalisé spécialement le module GML pour la gestion de la mémoire libre utilisée par XCP.

A l'initialisation de XCP, la totalité de la mémoire libre nécessaire à XCP est demandée au module STORAGE par GML et est rendue en fin de session. Les demandes et les libérations de mémoire nécessaire au fonctionnement de XCP sont satisfaites par le module GML qui gère la zone de mémoire libre initiale. Au cours du fonctionnement de XCP, la mémoire de la machine virtuelle est ainsi structurée :

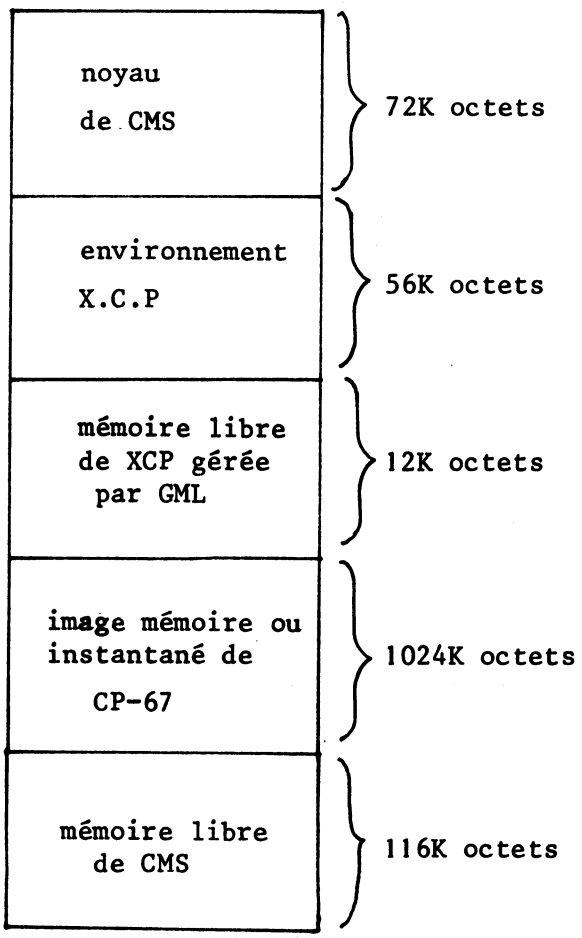


figure 4.7

Organisation de la mémoire

Remarque

Les tailles des trois dernières zones sont données à titre indicatif, car elles dépendent de la taille de XCP et de la taille mémoire du 360/67 réel.

4.1.4.2 Gestion du 2250

La gestion de l'unité 2250 se fait à 2 niveaux. Le premier niveau (niveau physique) est assuré par le module IO2250 et le deuxième niveau (niveau logique) par le module CONSOL. La circulation des informations est schématisée par la figure 4.8.

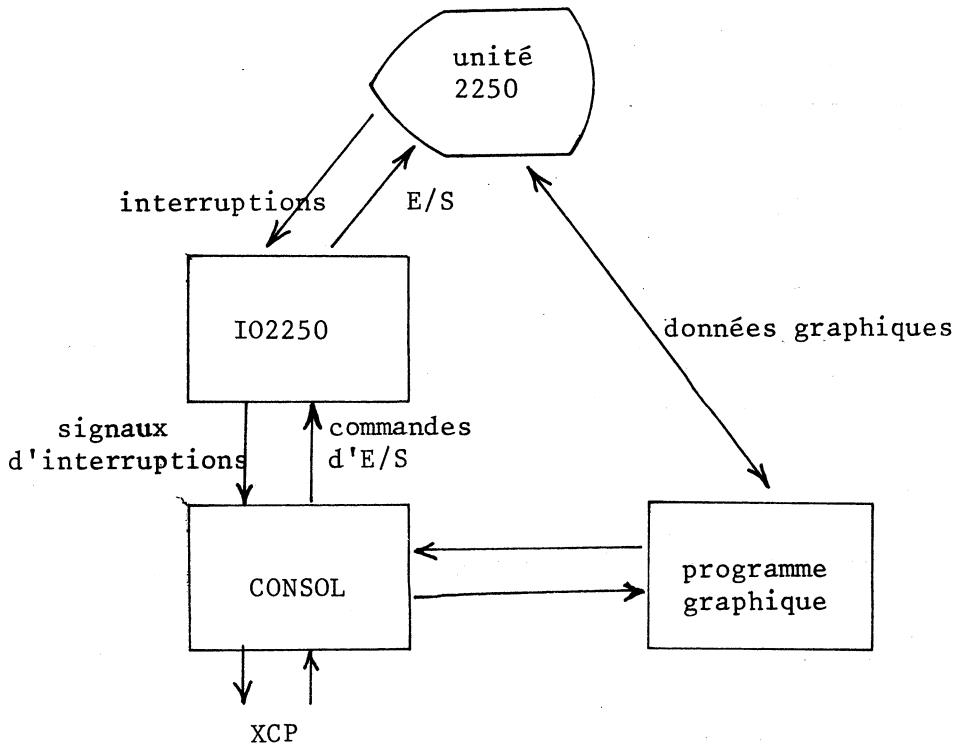


figure 4.8

Entrées/sorties sur le terminal graphique

Module IO2250 : Ce module est responsable de la gestion des entrées/sorties physiques et des interruptions.

Interruptions : lorsqu'une interruption de l'unité 2250 se produit, IO2250 reçoit le contrôle et analyse l'interruption. Toutes les informations utiles sont mises dans une zone de mémoire pour être transmises au module CONSOL. Le rôle de IO2250 est également de faire une pile des interruptions successives pour les restituer une à une au module CONSOL.

Demande d'entrée/sortie : IO2250 reçoit une demande d'entrée/sortie par un appel spécifique à CMS [Ref.15], accompagné d'un programme-canal. IO2250 analyse la demande et réalise toutes les entrées/sorties physiques.

Module CONSOL : le rôle de CONSOL est de réagir aux signaux d'interruptions en provenance du module IO2250 et d'émettre les demandes d'entrées/sorties.

Traitement des interruptions

Les interruptions du 2250 peuvent se classer dans 3 catégories :

- Clavier alphanumérique (caractères EOB et CANCEL).
  Lorsque ces signaux sont émis , CONSOL active le sous-programme qui vérifie la validité de l'information reçue.

- Touches des fonctions programmées
  Une interruption provoquée par une touche entraine plusieurs actions du module CONSOL : vérification de la validité (touche allumée ou éteinte), vérification de l'existence du programme correspondant et enfin activation de ce dernier.

- Pointeur optique
  Une correction est appliquée systématiquement sur les coordonnées reçues, car elles ne correspondent pas exactement à celles du caractère désigné par le pointeur optique [Ref.13].

En fonction du mode d'utilisation du pointeur optique (ADDRESS ou POINTER), différents sous-programmes sont ensuite appelés pour insérer un curseur, reconnaître le mot-clé désigné, etc...

Demandes d'entrée/sortie : Lorsque CONSOL reçoit une demande de lecture, il active le sous-programme responsable de la gestion de la zone d'entrée. Une requête est prélevée dans cette zone ou si cette dernière est vide, le sous-programme se met en attente jusqu'à la prochaine interruption en provenance du clavier alphanumérique.

Une demande de sortie correspond simplement à l'exécution d'un programme-canal d'écriture d'une ligne sur l'écran. Avant cette écriture, plusieurs actions sont effectuées pour :

- déterminer dans quelle zone de l'écran, l'écriture doit se faire

- remettre à blanc une partie de l'écran, ou décaler des lignes selon le mode de fonctionnement de l'écran (mode ROLL ou PAGE)

- calculer l'adresse de la ligne à écrire dans la mémoire d'entretien de l'unité 2250.

Programme graphique : Une copie du programme graphique propre au 2250 est constamment gardée en mémoire centrale. Elle contribue à l'initialisation des images correspondantes aux sous-environnements et sert aussi de référence pendant le traitement de certaines interruptions "graphiques". Le programme graphique est constitué d'ordres et de données graphiques dont la génération est faite à l'aide des macro-instructions GPS du système OS/360 [Ref.18].

4.2 DESCRIPTION DES DONNEES

Pour le système XCP, l'utilisation des variables se fait en deux étapes distinctes :

1/ La première étape consiste en une lecture des définitions symboliques telles que l'utilisateur les donne. Cette forme de description est purement formelle et n'est pas directement utilisable par les divers composants de XCP.

2/ La deuxième étape comporte une génération de différentes définitions en un code interne et une construction de liens entre les différentes tables pour faciliter le fonctionnement des divers modules de XCP.

Ces deux opérations sont exécutées chaque fois que XCP est initialisé.

4.2.1 Représentation interne

Une variable valide possède une représentation interne et est accessible à tout instant. Cette représentation est gardée en mémoire jusqu'à la fin de la session, à moins que l'utilisateur la supprime au moyen d'une requête SUPPRESS.

Les variables sont organisées en liste, ordonnée alphabétiquement. Cette liste est repérée par la variable interne SYMBOLS (fig.4.9).

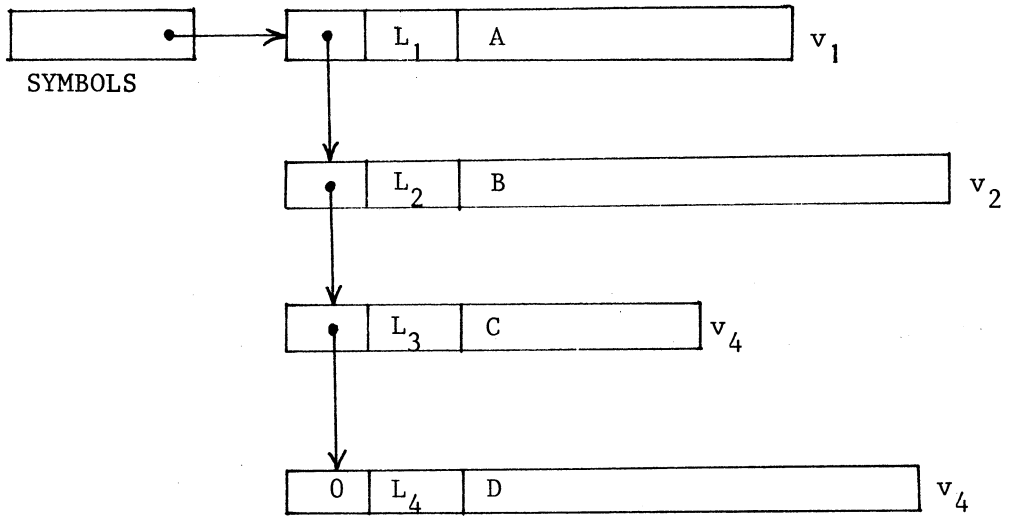
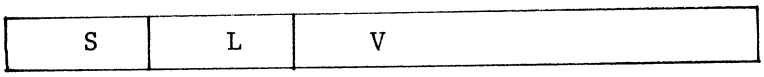


figure 4.9

Liste des variables

4.2.1.1 Représentation d'une variable simple

Une variable simple est représentée par un seul élément de la liste des variables.



Cet élément comporte 3 parties :

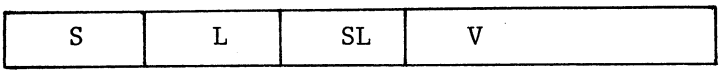
- S désigne le successeur de l'élément, ou zéro si l'élément est le dernier de la liste

- L longueur en octets de la partie V

- V chaîne de caractères donnant la représentation externe de la variable.

4.2.1.2 Représentation d'une variable structurée

Les variables structurées sont, chacune, représentées par une sous-liste dont l'en-tête appartient à la liste des variables repérées par SYMBOLS. L'en-tête d'une variable structurée comporte 4 parties :



- S successeur de l'élément dans la liste des variables
- L longueur de la partie V
- SL pointeur sur la sous-liste représentant les éléments de la structure
- V chaîne de caractères donnant la représentation externe de l'en-tête de la structure.

Les variables de la structure sont représentées soit par un élément décrit dans le paragraphe 4.2.1.1 si la variable est une variable simple, soit par une sous-liste si la variable est à nouveau une structure. (fig.4.10).

Remarques

La partie pointeur du dernier élément de la structure n'est pas nulle, mais repère l'en-tête de la structure correspondante

La figure 4.10 montre la représentation d'une structure à 2 niveaux.

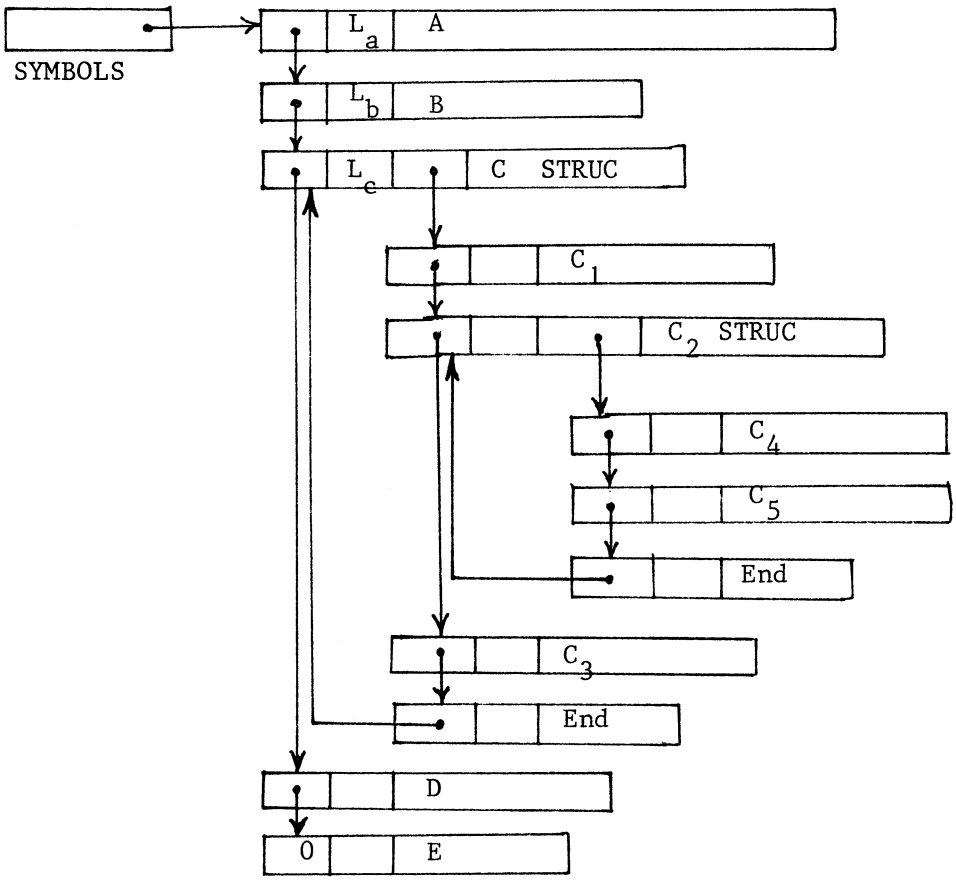


figure 4.10

Représentation des variables structurées

La partie L de l'en-tête ou de la fin d'une structure (ou sous-structure) contient en plus de la longueur :

- un indicateur de début ou de fin de structure suivant le cas
- le numéro du niveau de structure ou sous-structure

4.2.1.3 Construction de la liste des variables

La liste des variables (fig.4.10) est construite à chaque initialisation de XCP, à partir des fichiers sauvegardés au cours des

sessions précédentes par les requêtes SAVE ou ENDEF. Cette liste est mise à jour chaque fois qu'une modification est apportée à une variable (requêtes MODIFY et SUPPRESS) ou qu'une définition nouvelle apparaît (requêtes DEFINE).

L'adjonction des nouvelles variables se fait selon l'organigramme donné ci-dessous (fig.4.11).

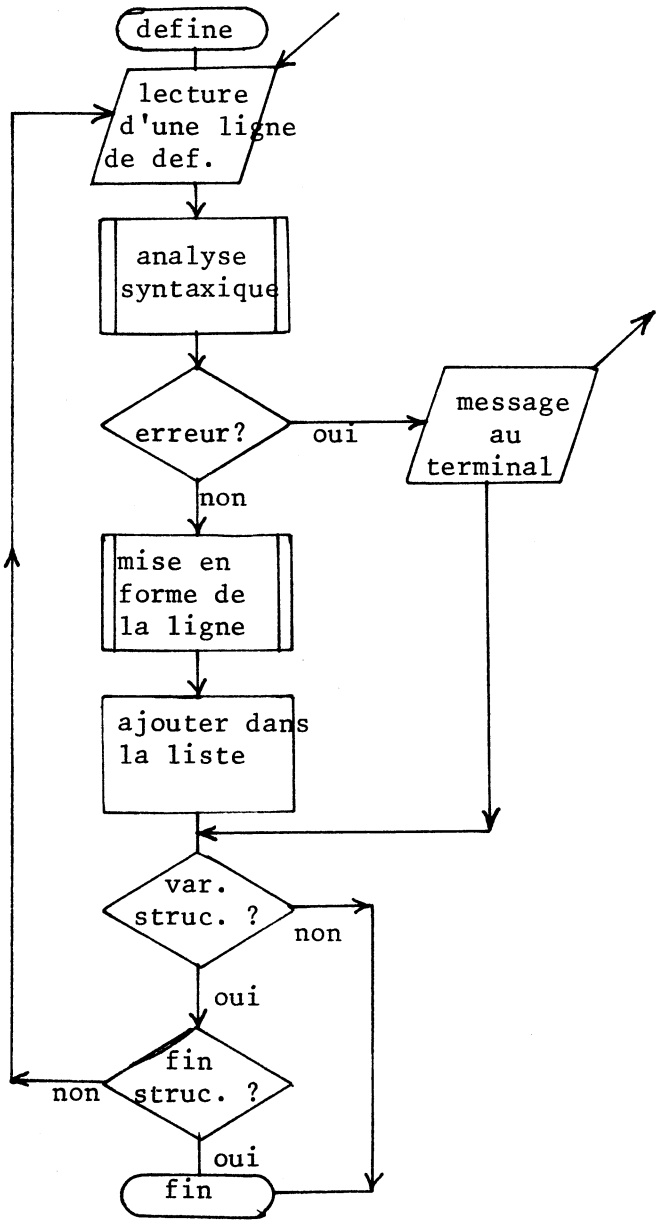


figure 4.11
Insertion des définitions

L'analyseur syntaxique contrôle la syntaxe de la définition, puis, si la syntaxe est correcte, la met en forme adéquate et la chaîne à la liste existante. L'analyse syntaxique est faite par le module CHECKER suivant le principe de fonctionnement d'un automate d'états finis. Au niveau de la définition, aucune vérification n'est faite sur l'existence des variables référencées dans les variables de type REF et STRUC, ou dans une **BASE** symbolique. La vérification est effective lors de la génération des descripteurs internes, ainsi que des vérifications sémantiques nécessaires.

4.2.2 GENERATION DES DESCRIPTEURS INTERNES

La représentation symbolique décrite précédemment ne peut être confiée aux modules de l'environnement XCP, car certaines informations ne sont pas explicites (liens), vérifiées (références) converties (nombre) etc... Un deuxième traitement est donc nécessaire; il est fait à partir de la représentation symbolique interne et consiste à générer pour chaque variable, un descripteur (VDCT) donnant toutes les caractéristiques de celle-ci. Cette opération n'est possible que lorsque l'utilisateur a terminé ses déclarations, c'est-à-dire à l'émission de la requête ENDEF. Les VDCT ne peuvent, en effet être générés au fur et à mesure des définitions lues compte tenu des liens (références) possibles entre les différentes variables

4.2.2.1 Contenu d'un VDCT

Chaque descripteur de variable VDCT est constitué de 4 double-mots (fig.4.12) dont les différents champs sont :

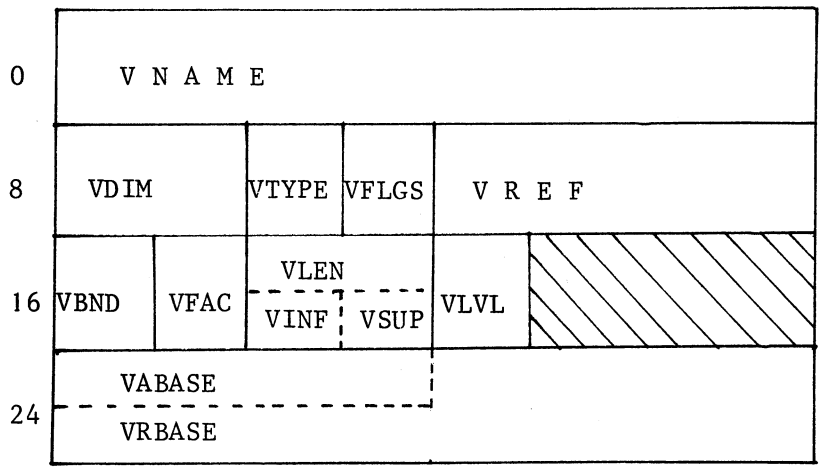


Figure 4.12

Contenu d'un VDCT

VNAME : contient une chaîne de caractères représentant le nom de la variable décrite.

VDIM : contient le nombre d'éléments de la variable dans le cas d'un tableau, sinon zéro.

VTYPE : contient un nombre représentant le type de la variable.

VFLGS : contient des indicateurs binaires pour usage interne.

VREF : ce champ n'est utilisé que pour les variables de type STRUC ou REF. Dans le cas de STRUC, il contient un pointeur vers le vecteur de la structure (fig.4.13) et pour REF un pointeur vers le VDCT de la variable référencée.

VBND : contient des indicateurs binaires pour les longueurs symboliques. Ils servent à signaler une spécification d'alignement (double mot, mot et demi mot)

VFAC : contient le facteur de duplication dans le cas des longueurs symboliques.

VLEN : contient la longueur en octet de la variable (ou d'un élément dans le cas d'un tableau)
Pour une variable BITST ou FIELD, ce champ se subdivise en 2 parties :

VINF contient le numéro du bit représentant la borne inférieure de la chaine de bit BITST ou du champ FIELD

VSUP contient le numéro du bit représentant la borne supérieure de la chaîne de bit BITST ou du champ FIELD.

VLVL : contient le niveau de la variable. Toutes les variables simples ont le niveau zéro.

VABASE : contient l'adresse absolue de la variable ou dans le cas des variables de niveau supérieur à zéro, l'adresse relative (OFFSET) par rapport à la première variable de la structure ou sous-structure.

VRBASE : contient une chaîne de caractère représentant le nom de la base.

4.2.2.2 Vecteurs de VDCT

Tous les descripteurs de même niveau sont regroupés dans une table appelée vecteur de VDCT. Les champs VREF des structures repèrent les vecteurs des variables de niveau inférieur , constituant ainsi une structure arborescente (fig.4.13).

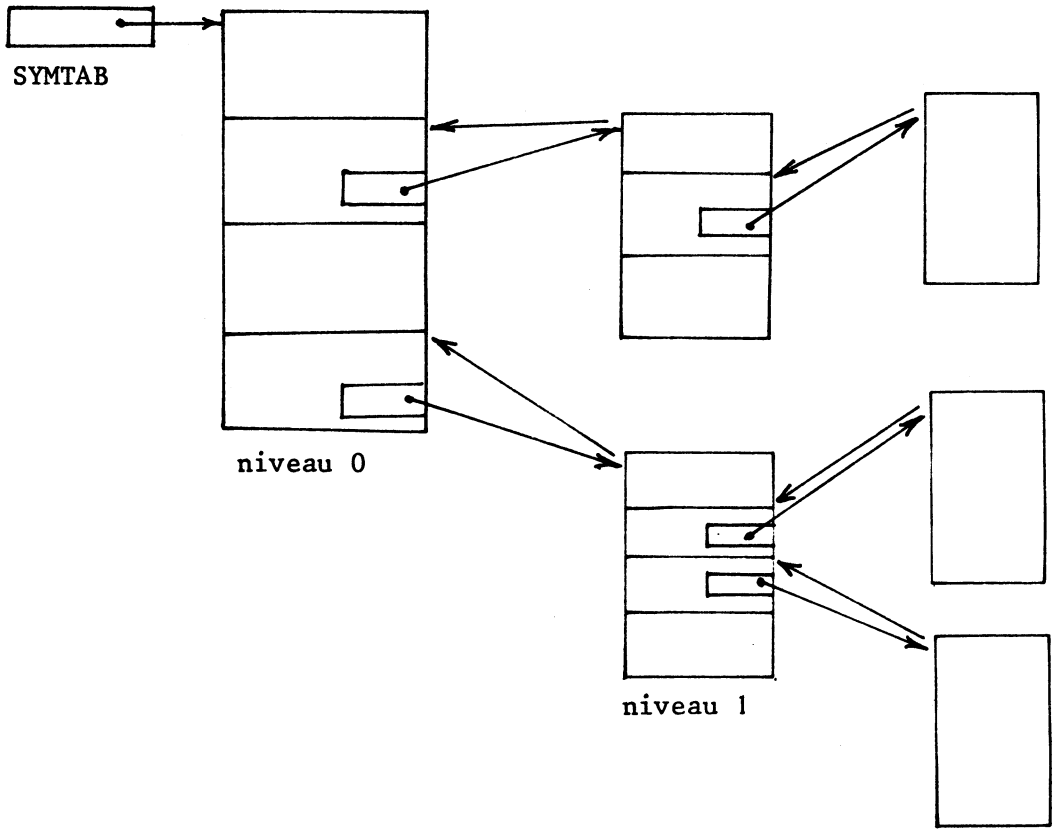


Figure 4.13

Relations entre les vecteurs de VDCT

En tête de chaque vecteur nous trouvons des informations telles que : l'adresse du VDCT dont le vecteur est le constituant, et le nombre d'éléments du vecteur.

Remarque

> Les liens de type REF ne sont représentés sur la figure 4.13 pour ne pas alourdir celle-ci.

4.2.2.3 Vérifications et liens

Au cours de la génération des VDCT et des vecteurs de VDCT,

diverses vérifications sont faites sur la validité du contenu de la représentation interne, par exemple : le nombre d'éléments, la longueur, l'alignement, etc... Les erreurs éventuelles sont signalées et peuvent selon leur gravité arrêter le processus de génération.

Lorsque tous les vecteurs sont construits, les différents liens sont établis et des erreurs de référence indéfinie peuvent alors apparaître mais elles n'arrêtent pas la génération. Une référence non résolue empêche ultérieurement l'exécution qui utilise cette information.

Remarque

> Il n'existe pas de lien explicite entre la liste symbolique et les VDCT.

4.3 ANALYSE D'UN INSTANTANE

L'examen d'une image-mémoire ou d'un instantané de CP-67 de façon symbolique fait intervenir des mécanismes particuliers soit au niveau des requêtes de base soit au niveau des requêtes du langage d'extension. Dans les paragraphes suivants, nous dégagerons quelques idées sur les mécanismes utilisés.

4.3.1 Mécanismes de base

Pour réaliser leurs fonctions, les requêtes de base font appel à des mécanismes internes utilisant des structures de données, des algorithmes particuliers, etc... Nous examinerons comme exemples les

mécanismes concernant les requêtes SNAP, QUALIFY et REMEMBER.

4.3.1.1. Acquisition d'une image-mémoire ou d'un instantané (requête SNAP)

a) La lecture d'une image-mémoire est simple puisqu'il s'agit tout simplement d'une transfert de données depuis les mémoires secondaires (bande, disque, fichier de "spooling") vers la mémoire centrale. Le transfert de l'image-mémoire a lieu lorsque la requête ENDEF est émise.

b) L'acquisition d'un instantané pose certains problèmes, étant donnée la nature des informations. En effet, CP-67 doit être capable de transférer sa propre mémoire vers la mémoire de la machine virtuelle faisant fonctionner XCP. Cette opération doit obéir à certaines conditions, à savoir :

- ne pas perturber le fonctionnement des autres machines virtuelles actives

- donner un instantané dont le contenu est cohérent, c'est-à-dire que le transfert ne doit pas être gêné ou interrompu par d'autres processus capables de modifier l'état de la mémoire centrale réelle.

Nous avons résolu le problème de deux façons suivantes :

1/ Transfert mémoire à mémoire

La copie de la mémoire réelle dans la mémoire virtuelle se fait par le mécanisme de pagination de CP-67. Pour déclencher l'opération, l'instruction spéciale de communication machine virtuelle - CP-67 DIAGNOSE [Ref.14] est exécutée par XCP, ce qui signale au système CP-67 qu'une opération de transfert doit avoir lieu. CP-67 recopie alors

toute sa mémoire (c'est l'instantané) sur les unités à accès direct utilisées pour la pagination (tambour et/ou disque). Les adresses des enregistrements (adresse unité, numéro cylindre etc...) sont gardées dans la table de "swapping" [Ref.14, 28] de la machine virtuelle, et les pages "swappées" sont marquées non en mémoire réelle. Lorsque XCP fera par la suite référence aux pages copiées, les modules de pagination de CP-67 amèneront les enregistrements correspondants en mémoire virtuelle de XCP.

Au niveau du système CP-67, la seule précaution à prendre pour réaliser ce mécanisme est d'exécuter la recopie des pages de façon autonome et non interruptible.

2/ Transfert mémoire-bande

La deuxième solution adoptée est l'utilisation d'une bande magnétique comme intermédiaire entre CP-67 et la machine virtuelle. Le déclenchement de l'opération est toujours commandé par une instruction DIAGNOSE.

Dès la réception de la demande, CP-67 effectue la copie de la mémoire réelle sur la bande magnétique spécifiée. Le contrôle est ensuite redonné à XCP qui lit cette bande magnétique pour amener l'instantané en mémoire virtuelle.

Cette méthode de transfert a deux avantages sur la précédente: possibilité d'archivages des instantanés et une programmation plus simple des opérations. Mais par contre, elle présente l'inconvénient important de bloquer toutes les machines virtuelles pendant un laps de temps relativement long (15 secondes environ). Ceci est dû à la non simultaneité des opérations d'entrée-sortie et de la lenteur relative des unités de bandes magnétiques.

4.3.1.2 Qualification des variables structurées (requête QUALIFY).

Dans le chapitre précédent, nous avons vu que pour accéder aux variables d'une structure, une "qualification" de cette dernière est nécessaire. Cette opération est équivalente à :

- spécifier le domaine d'action des requêtes, c'est-à-dire restreindre les variables accessibles

- décrire le cheminement choisi à travers les différents chemins possibles, afin de supprimer les ambiguités.

Une structure"qualifiée" est représentée en mémoire par un élément d'une pile, dont le sommet désigne la structure la plus récemment qualifiée (fig.4.14).

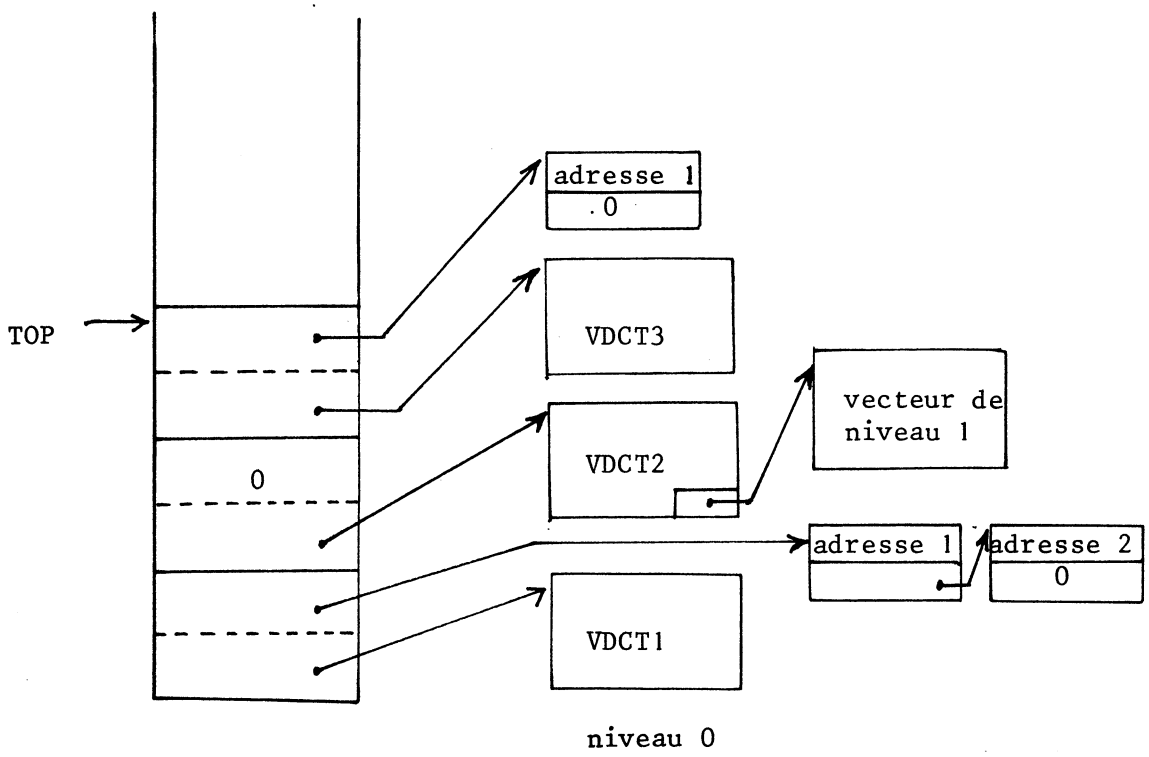


figure 4.14

Pile des structures "qualifiées"

A un instant donné, seules les variables appartenant à la structure du sommet de la pile sont accessibles. De ce fait, une requête QUALIFY conduit à ajouter un élément dans cette pile. Chaque élément de la pile est constitué de 2 pointeurs :

- un pointeur vers le VDCT décrivant la structure "qualifiée"

- un pointeur vers la liste des adresses successives des occurrences (§ paragraphe 3.2.1.3.2 c) et d)) de cette structure. Les requêtes NEXT et BACK ajoutent ou enlèvent un élément de cette liste. Lorsque la structure qualifiée ne représente pas un élément de liste, ce dernier pointeur est nul.

Grâce à cette pile, le cheminement défini par l'utilisateur à travers la structure est connu et peut être conservé par une requête REMEMBER.

4.3.1.3 Activation des chemins (requête REMEMBER)

Pour conserver un chemin (contexte) donné, il suffit de recopier l'état de la pile au moment où la requête REMEMBER est émise. Cette opération ajoute dans la liste repérée par REMLIST, la réplique exacte de la pile de qualification, avec la liste des adresses de chaque élément de la pile (fig.4.15).

Chaque élément de la liste repérée par REMLIST est appelé REMBLOK et contient les champs :

REMNEXT pointeur sur le successeur
REMSTAK pointeur sur le sommet de la pile copiée
REMLAST pointeur sur le bas de la pile copiée
REMNAME nom du chemin donné par l'utilisateur

Avec cette organisation, l'activation d'un chemin revient alors à prendre en compte l'élément nommé de la liste des chemins sauvegardés et de reconstruire la pile de qualification.

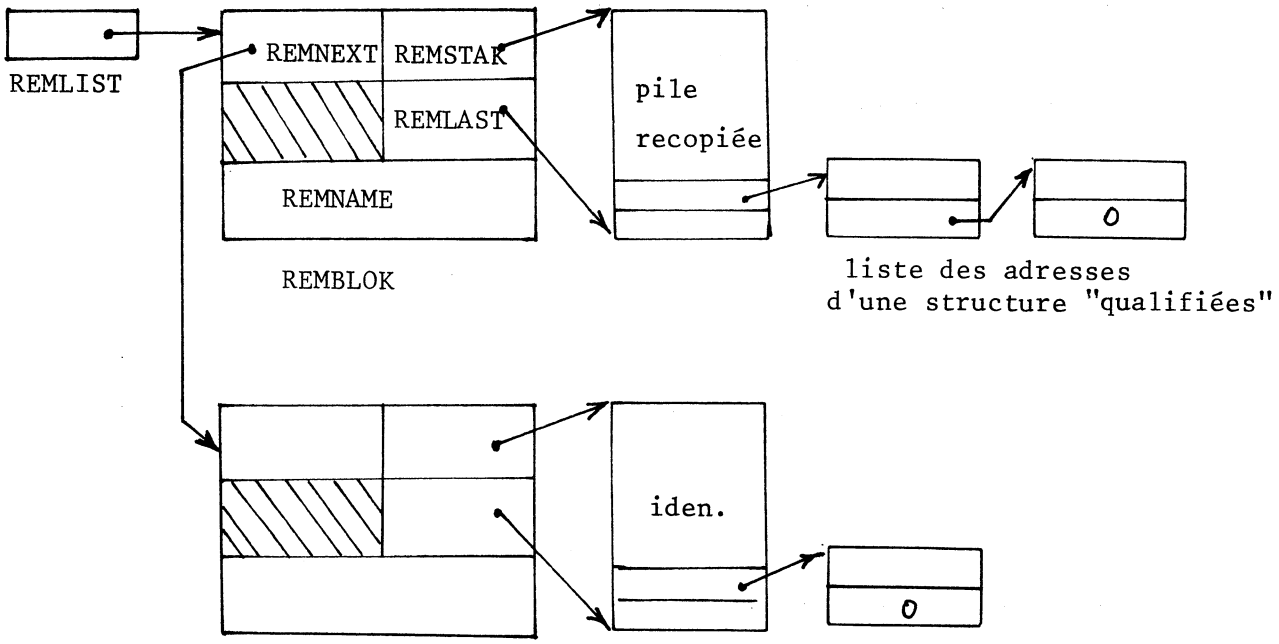


figure 4.15

Liste des chemins conservés

4.3.2 Mécanisme des macro-requêtes

Selon l'organisation adoptée (fig.4.1) pour XCP le macro-processeur est en amont du processeur des requêtes de base, et peut donc analyser chaque requête entrée depuis le terminal. Cette analyse est faite par le module FILTER qui active soit le processeur soit le macro-processeur. Nous n'examinerons pas le cas de l'appel du processeur des requêtes de base mais celui du macro-processeur MACPROC; il reçoit le contrôle lorsqu'il y a demande de traitement pour :

- une expression d'affectation d'une variable globale

- une demande d'impression de la valeur d'une variable globale

- une appel d'une macro-requête.

Les deux premiers cas correspondent à l'utilisation de XCP comme machine de bureau. Une affectation de valeur est reconnue par la présence d'une variable globale (&G0,...,&G9) et du caractère =, suivi de l'expression à évaluer. Le contrôle est alors donné au module EVALUATE, qui analyse l'expression arithmétique et calcule sa valeur. La demande d'impression de la valeur d'une variable globale, quant à elle, est reconnue par la présence seule du nom de la variable. Cette possibilité de XCP exige par conséquent que les noms des macro-requêtes ne soient pas les mêmes que ceux des variables globales. Quant à l'exécution d'une macro-requête, elle comprend 2 phases :

1/ Phase d'initialisation

Lorsqu'une macro-requête doit être exécutée, cette phase demande la zone de travail nécessaire à GML (environ 1K octets) pour

ranger les paramètres d'appel, les variables locales, etc... propre à chaque niveau récursivité. Celui-ci est augmenté de 1 chaque fois que cette phase est exécutée. Cette phase vérifie l'existence du fichier contenant la macro-définition et la validité des différents paramètres d'appel. Ceux-ci sont reconnus uniquement par leur position.

2/ Phase d'exécution

C'est elle qui contrôle le déroulement des opérations provenant des requêtes de la macro-définition. 3 cas peuvent se présenter :

1) la requête est une affectation d'une valeur à une variable arithmétique ou caractère :
   Pour une variable arithmétique (&GO, &G9, &IO,...&I9) l'évaluation de l'expression est faite comme précédemment par le module EVALUATE.
   Pour une variable caractère, le sous-programme KEYWCK est appelé pour rechercher sa valeur, ou créer la variable. Cette opération est faite à partir d'une liste, repérée par KEYWORDS (fig.4.16), locale à chaque niveau de récursivité. Elle n'est créée que lorsqu'il y a utilisation d'une variable de type caractère.

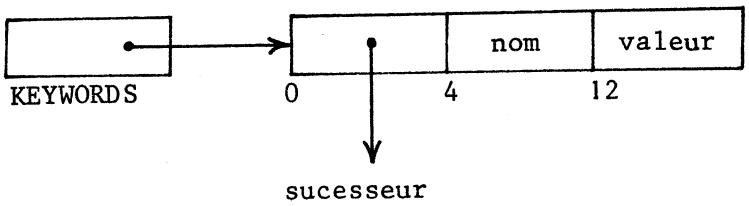


Figure 4.16

Représentation d'une variable caractère

2) la requête n'est pas une affectation: les paramètres d'appels sont substitués aux paramètres donnés dans la définition, puis une décision est prise quant à la destination de la requête ainsi traitée :

- le premier mot de la requête n'est pas préfixé par le caractère &. La requête est alors dirigée vers le processeur des requêtes de base.

- Le premier mot n'est pas un mot-clé connu de MACPROC. Un appel (récursif) à MACPROC est fait, avec comme paramètre, la requête analysée.

- Le premier mot est un mot-clé connu de MACPROC. La requête appartient au langage d'extension et elle est alors dirigée vers les différents sous-programmes chargés d'analyser son contenu puis de les interpréter.

3) Lecture au terminal d'une ou des requêtes par &READ

C H A P I T R E 5

CONCLUSION

La réalisation de l'environnement d'aide XCP a été rendue aisée par la simplicité des structures internes du système CP-67 et des objectifs que nous nous sommes fixés. Cependant pour ne pas alourdir la programmation, certaines restrictions sont imposées dans les 3 aspects de l'outil : langage de description, langage des requêtes et langage d'extension.

Nous explicitons brièvement les limitations et les problèmes posés par ces langages.

Langage de description

La principale difficulté réside dans la conception du langage puisqu'il doit décrire un ensemble de structures de données déjà existant. En effet, il est beaucoup plus aisé de décrire des données à partir d'un langage plutôt que de définir un langage devant s'adapter à une structure particulière. Notre langage de description permet de décrire la quasi-totalité des blocs de contrôle du système CP-67. Seuls quelques blocs restent en marge des possibilités du langage; ceci est dû aux particularités de CP-67 ou à certains problèmes de réalisation de XCP.

Exemples

- la taille des tableaux est statique et doit être connue au moment de la description. En effet cette information n'est pas contenue dans les données et il est donc impossible à à XCP de la déterminer par un algorithme quelconque. L'exemple le plus typique est le bloc de contrôle LOADMAP (carte de chargement des modules du système CP-67).

- Les zones de mémoire libre ont une longueur statique pour les mêmes raisons que les tableaux.

La figure 5.1 montre la représentation d'un bloc de mémoire libre gérée par CP-67. Dans le premier mot se trouve l'adresse du successeur et dans le second, la longueur totale du bloc exprimée en double-mot.

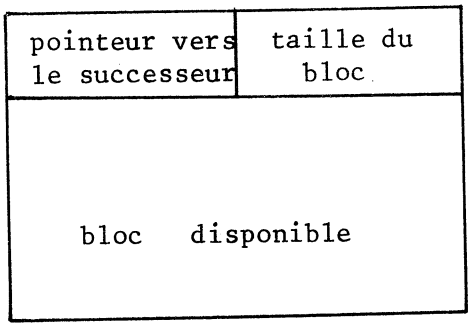


Figure 5.1

Bloc de mémoire libre

Pour donner la taille d'un bloc au moment de la description il faudrait spécifier que : "la taille de ce bloc est exprimée en double-mot et qu'elle se trouve à 1'adresse relative 4 du bloc"! Exprimer cette information dans un langage de description est à notre avis difficile.

Les références à une variable ne peuvent être que des identificateurs et non des expressions. Ceci ne nous a pas facilité la description des relations entre les tables des segments, les tables pages et les tables de "swapping" (fig.5.2)

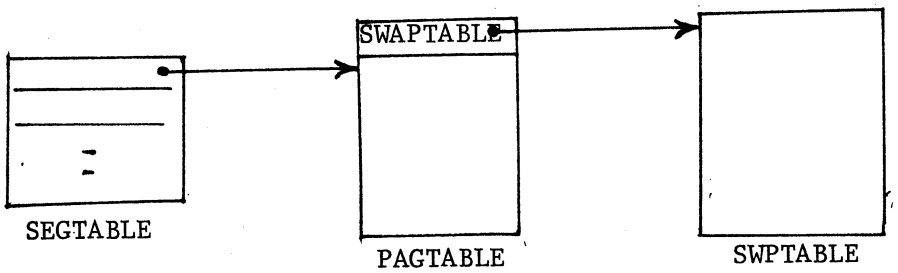


Figure 5.2

Les relations entre les tables des segments des pages et de swapping

En effet chaque entrée d'une SEGTABLE pointe sur le deuxième mot de la PAGTABLE. La relation entre ces deux tables s'exprimerait alors par :

REF(PAGTABLE+4)

Requêtes de base

Au niveau des requêtes, 2 remarques peuvent être faites en ce qui concerne leur nombre et la qualification des variables référencées.

Les requêtes sont en effet en nombre limité car nous avons voulu créer un noyau minimal extensible. L'extension des requêtes de base peut se faire très facilement à l'aide des macro-requêtes ou par l'implantation des nouvelles requêtes lorsque les besoins exacts sont définis. Cette dernière possibilité est facile à mettre en oeuvre étant donné l'organisation adoptée par le processeur des requêtes.

La qualification des variables structurées est obligatoire pour simplifier la structure interne de XCP. Nous nous sommes refusés à créer un outil trop complexe donc par la même occasion moins souple et moins bien organisé.

Langage d'extension

Notre travail nous a mené à développer un macro-processeur spécialement pour l'environnement XCP car celui du système CMS ne nous convient pas. Ce besoin se fait sentir chaque fois que sont créés des outils conversationnels (éditeur, metteur au point, etc...) mais peut être évité en donnant au macro-processeur du système quelques caractéristiques telles que :

- Le macro-processeur du système doit être accessible à tous les processeurs logiques du système, quel que soit le niveau où ils se trouvent. Tous les environnements conversationnels du système seraient alors extensibles sans problème et de façon unique

- Les concepts et les caractéristiques du macro-processeur doivent être clairement définis (substitution des variables, types de données, liens avec les différents processeurs, etc..)

- L'exécution des macro-requêtes peut encore être améliorée car actuellement l'analyse des macro-requêtes se fait de façon interprétative à partir des fichiers sur disque.

Quant aux développements et à la continuité de notre travail, mis à part le macro-processeur , nous insistons surtout sur :

1) L'extension du langage de description :

Il serait souhaitable d'avoir un langage unique qui servirait à la fois à la conception et à la maintenance du système. Le langage de description doit posséder entre autres les caractéristiques telles que :

- la description des structures de données doit être séparée de la description des algorithmes opérant sur ces structures, afin de pouvoir modifier séparément les deux parties de façon indépendante.

- Les formes particulières des données doivent être facilement décrites et facilement accessibles (piles, anneaux, liste, etc...)

- La définition des structures de données doit pouvoir être "conditionnelle", c'est-à-dire que leur existence doit être liée à une condition provenant de la valeur d'une autre variable. Ces cas sont

fréquemment rencontrés dans les systèmes; en particulier dans le bloc de contrôle DCB de OS/360 , dans l'extension de la UTABLE d'une machine virtuelle 360/67 dans CP-67.

- Une spécification des valeurs, ou d'une plage de valeurs pouvant être prise pour une variable, pour permettre un contrôle pendant la marche du système ou au cours de l'examen de la mémoire.

Ces notions reflètent la tendance des langages de programmation système actuels [Ref.2,4,38] ou des particularités des nouveaux softwares [Ref.1].

2) L'outil d'enseignement d'un système:

Beaucoup de choses restent encore à faire dans ce domaine afin d'éviter que l'enseignement d'un système d'exploitation ne se résume à une revue systématique de tous les formats des blocs de contrôle et l'examen d'une centaine de pages de dump .

Il serait intéressant de développer également un système question-réponse qui à l'aide de la description symbolique fournie, puisse donner des renseignements sur les structures des données.

3) L'adaptation de XCP aux autres systèmes:

En fonction des résultats et de l'expérience acquis avec l'utilisation de XCP, nous pouvons alors envisager l'adaptation de notre environnement aux systèmes existants. Certaines modifications seront certainement nécessaires surtout en ce qui concerne le langage de description mais aussi les mécanismes de base, pour adapter l'outil aux particularités des systèmes à analyser.

4) La machine virtuelle de diagnostic "on-line" :

Nous pensons également faire de XCP, un outil de diagnostic pendant le fonctionnement de CP-67. Le programmeur système aura en permanence tous les renseignements voulus sur le comportement du système et pourra modifier dynamiquement les différents blocs de contrôle grâce aux macro-requêtes, il pourra créer de nouvelles fonctions de CP-67, tout en étant sur une machine virtuelle. L'environnement XCP sera ainsi considéré comme une extension de CP-67, sans que les instructions correspondantes appartiennent au noyau résident du système.

Grâce à cette possibilité, des interventions "à chaud" pourront être faites pour dépanner le système, évitant ainsi les inconvénients inhérents à un redémarrage de celui-ci (perte d'informations, perte de temps, etc...). Toutefois de nombreux problèmes restent à résoudre, en particulier celui de la validité et de la cohérence des données acquises dynamiquement. En effet, pendant qu'une donnée est analysée par l'utilisateur, des changements peuvent avoir lieu dans les structures de CP-67, et les modifications apportées risquent de se faire sur une donnée différente de celle analysée.

Pour la réalisation des systèmes futurs nous pensons qu'il serait souhaitable de poser à priori le problème de la maintenance, les corrections des erreurs, de la mise au point avant la mise en oeuvre du système, car à mesure que le projet avance, il est difficile de modifier ses caractéristiques pour résoudre les problèmes cités. Quelques réalisations tant du côté hardware que du côté software, nous amènent à conclure que ces problèmes ne sont pas du tout négligés :

- l'apparition des dispositifs technologiques destinés à faciliter la mise au point des programmes (Program Event Recording) [Ref.24].

- l'utilisation des langages spécialisés et de la programmation structurée dans la conception des systèmes [Ref.2,4,34].

A N N E X E

MANUEL D'UTILISATION DE XCP

L'environnement d'aide XCP est un ensemble de programmes conçu pour fonctionner sur une machine virtuelle gérée par le système CMS. Il peut être initialisé par la requête :

XCP ([TYPE] [{ DISK / TAPE / RPRT }])

TYPE indique qu'un terminal classique est utilisé (IBM 2741 ou TTY). Si cette option est absente, le terminal est supposé être un terminal graphique IBM 2250 avec les dispositifs crayon optique et clés de fonctions programmées.

{ DISK / TAPE / RPRT } L'image-mémoire à examiner se trouve respectivement sur disque, bande magnétique ou un remote-printer (unité virtuelle spéciale). Si aucune de ces 3 options n'est présente, XCP suppose que l'instantané sera obtenu au cours de la session par la requête SNAP.

Dans les pages qui suivent nous donnons par ordre alphabétique la liste de toutes les requêtes du système XCP. Le fonctionnement de chacune d'elles est expliqué, ainsi que les différents paramètres requêtes.

Les conventions suivantes sont utilisées pour détailler chaque requête.

ALL Lorsqu'un paramètre est donné en majuscule, il s'agit d'un mot clé et doit être utilisé comme tel.

<nom> Le paramètre doit être remplacé par sa valeur, par exemple un identificateur, un nombre décimal ou hexadécimal, un opérateur arithmétique, etc...

[ALL] Les crochets indiquent que le paramètre est facultatif donc qu'il peut être omis.

{ ALL / <nom> } Les accolades indiquent qu'il y a un choix entre différents paramètres

Les détails concernant la génération de XCP et la définition de la machine virtuelle sur laquelle doit fonctionner XCP, sont donnés dans le chapitre 4.

Requête ACTIVATE (Sous-environnement d'analyse)

Format : ACTIVATE < nom >

Utilisation : Un chemin (contexte) mémorisé par la requête REMEMBER peut être rendu actif par la requête ACTIVATE. Toutes les structures "qualifiées" courantes sont automatiquement rendues inactives, avant l'activation du chemin demandé.

Messages : PATH NOT FOUND
Le chemin spécifié ne correspond à aucun chemin mémorisé.

Requête BACK (Sous-environnement d'analyse)

Format : BACK [< nombre >]

Utilisation : Lorsque la structure "qualifiée" est sous forme de liste, la requête BACK permet de se déplacer de n tables en arrière à partir de la table courante. Si < nombre > est omis, il sera pris égal à 1.

Messages : FIRST TABLE REACHED
La première occurrence de la structure est atteinte avant d'avoir épuisé les n occurrences spécifiées. La première occurrence est active.

NO QUALIFIED TABLE
La requête BACK est utilisée alors qu'une structure n'est pas "qualifiée" auparavant par une requête QUALIFY.

INVALID NUMBER
La syntaxe du nombre donné en paramètre est incorrecte.

NOT IN LIST FORM
La requête BACK ne peut être exécutée car la structure qualifiée n'est pas sous forme de liste.

Requête CALL (tous environnements)

Format : CALL < nom > [< parm1 > .....< parm >]

Utilisation : Cette requête permet d'appeler des programmes conçus de façon indépendante de XCP. Elle autorise ainsi une adaptation facile des programmes quelconques.

< nom > spécifie le nom du programme appelé

< parm > paramètres d'appel du programme repéré par le registre général 1 (la liste des paramètres est conforme aux listes de paramètres du système CMS).

Messages : PROGRAM DOES NOT EXIST
Aucun fichier de type TEXT correspondant au nom n'a été trouvé.

ERROR WHILE READING OBJECT CODE
Une erreur a été détectée durant la lecture du program objet.

ERROR WHILE LOADING PROGRAM
Une erreur a été détectée lors du chargement du programme obje

NO FREE STORAGE
Le programme spécifié ne peut être chargé en mémoire par manque de place.

Requête CP (tout environnement)

Format : CP < fonction console CP-67 >

Utilisation : Cette requête permet d'exécuter une fonction console de CP-67 sans quitter l'environnement XCP.

Message : Les messages sont variables et dépendent de la nature de la fonction console utilisée. (§ brochure CP-67/CMS [Ref.17])

Requête **CVT** (tout environnement)

Format : CVT < nombre >

Utilisation : Cette requête permet de faire la conversion d'un nombre hexadécimal et vice-versa. Les nombres à convertir sont des entiers non signés. Les nombres hexadécimaux doivent être préfixés du caractère # .

Messages : NUMBER TOO LARGE
Le nombre à convertir est trop grand. La limitation actuelle est de 8 chiffres pour les nombres décimaux et de 7 chiffres pour les nombres hexadécimaux.

INVALID NUMBER
La syntaxe du nombre à convertir est incorrecte.

Requête DEFINE (Sous-environnement de définition)

Format : DEFINE < définition de variable >

Utilisation : Cette requête doit être utilisée lorsque l'utilisateur veut rentrer de nouvelles définitions de variables simples ou structurées. Ces descriptions doivent obéir aux règles de syntaxe définies dans le chapitre 3.

< définition de variable > est la ou les lignes donnant la description de la variable.

Messages : ENTER STRUCTURE DEFINITION;
Ce message invite l'utilisateur à rentrer la suite de la définition d'une variable structurée. Lorsqu'une chaîne 'END' correspondant au niveau 1 de la structure est rencontrée la requête DEFINE se termine.

INCORRECT SYNTAX, RETYPE
La syntaxe n'est pas respectée, toute la ligne doit être retapée.

INVALID IDENTIFIER
L'identificateur d'une variable simple est "*". Ceci n'est autorisé que pour les variables appartenant à une structure.

ALREADY EXISTS
L'utilisateur a donné comme identificateur celui d'une variable déjà existante.

INVALID BIT DEFINITION
La variable de type BIT n'est pas précédée par une déclaration d'une variable de type BITST ou BIT.

INVALID FIELD DEFINITION
La variable de type FIELD n'est pas précédée par une déclaration d'une variable de type BYTE ou FIELD.

BASE ERREOR
Une base a été spécifiée dans une déclaration de variable appartenant à une structure.

BOUNDARY ERROR
Les spécifications de frontière (double mot, mot ou demi-mot) contenues dans la variable structurée ne peuvent être respectées.

Requête DISCARD (Sous-environnement d'analyse)

Format : DISCARD { ALL | [< nombre >] | UNTIL < nom structure > }

Utilisation : Cette requête rend inacccessible les structures "qualifiées" auparavant par la requête QUALIFY.

ALL : toutes les structures "qualifiées" sont rendues inactives.

<nombre> n structures "qualifiées" sont rendues inactives (à partir de la structure qualifiée la plus récente).

UNTIL < nom structure > rend les structures "qualifiées" inactives jusqu'à la structure < nom structure > non comprise.

Messages : NO QUALIFIED TABLE
Aucune structure n'a été "qualifiée" auparavant.

INVALID NUMBER
La syntaxe du nombre donné en paramètre est incorrecte.

UNKNOWN KEYWORD
Le mot-clé donné ne correspond ni à ALL ni à UNTIL

STRUCTURE NOT QUALIFIED
Le nom de la structure donné en paramètre ne correspond à aucune structure "qualifiée" précédemment. Aucune action n'est prise.

NO MORE QUALIFIED TABLE
Le nombre donné en paramètre dépasse le nombre de structures "qualifiées".

Requête DISPLAY (Sous-environnement d'analyse)

Format : DISPLAY < adresse > < type> (< longueur >)

Utilisation : Cette requête ne concerne que les variables ayant une base absolue. Elle permet de "voir" le contenu d'une zone traduite selon le < type > spécifié. <adresse>

< adresse > est la base absolue de la zone. Cette base peut être donnée sous forme de pointeurs.

< type > est le type choisi parmi les types valides BIT, BITST, BYTE, CHAR, DEC, INT et FIELD. CCW peut être également spécifié. Les données sont alors considérées et traduites en commande des canaux d'entrée-sortie.

< longueur > est la longueur de la zone. Elle peut être soit numérique soit symbolique avec facteur de duplication (§ Chapitre 3). Dans le cas où CCW est spécifié la longueur doit être numérique et elle indique le nombre de commandes à traduire.

Messages : INVALID ADDRESS, PAGE NOT AVAILABLE
L'adresse spécifiée en paramètre ne fait pas partie des pages de l'instantané. Seules les pages de CP-67 sont accessibles.

NULL POINTER
Dans la chaîne de pointeurs spécifiée, un pointeur nul a été trouvé.

BOUNDARY ERROR IN ADDRESS
L'adresse absolue de la zone ne respecte pas l'alignement spécifié par une longueur symbolique (D, W ou H).

OFFSET ERROR
Lorsque l'adressage avec les pointeurs est utilisé, l'adresse relative dans la chaîne des pointeurs n'est pas multiple d'une adresse de mot.

Requête DUMP (Sous-environnement d'analyse)

Format : DUMP { ALL | < adresse 1 > <adresse 2> [PRT] }

Utilisation : Cette requête imprime, soit sur le terminal soit sur l'imprimante (option PRT), le contenu d'une zone de mémoire sous forme de chiffres hexadécimaux et de caractères EBCDIC équivalents.

ALL imprime tous les registres et toute la mémoire sur l'imprimante.

<adresse 1> adresse de début de la zone à imprimer.

<adresse 2> adresse de fin de la zone à imprimer.

Messages : SNAPSHOT NOT AVAILABLE
Aucun instantané de CP-67 n'est accessible par XCP

ADDRESS ERROR
Une erreur est détectée dans les 2 adresses spécifiées

- <adresse 2> est inférieure à <adresse 1>
- la syntaxe des adresses n'est pas correcte
- la zone spécifiée ne se trouve pas dans l'instantané.

UNKNOWN KEYWORD
Les mots-clés donnés ne sont pas ALL ou PRT.

Requête ENDEF (Sous-environnement de définition)

Format : ENDEF [{ NOGEN / SKIP }] [SAVE]

Utilisation : Pour entrer dans l'environnement d'analyse l'utilisateur doit émettre la requête ENDEF.

NOGEN permet de ne pas générer les descripteurs internes des variables symboliques. Cette option économise de la place mémoire et du temps lorsque l'utilisateur veut uniquement travailler avec les adresses absolues.

SKIP permet d'ignorer les erreurs sémantiques et de lien lors de la génération des descripteurs internes. Si cette option est omise et si des erreurs sont détectées l'utilisateur reste toujours dans l'environnement de définition.

SAVE permet de sauver sur support externe toutes les nouvelles définitions de variables définies (sans erreur de syntaxe) lors de la session.

Messages : MAIN ENVIRONMENT...ENTER REQUEST
Signale à l'utilisateur de l'environnement d'analyse est actif.

SAVING...
Lorsque l'option SAVE est spécifiée la liste des variables structurées sauvées est imprimée à la suite de ce message.

UNRESOLVED REFERENCES
Les identificateurs référencés dans une variable de type REF ne sont pas définies.

BOUNDARY ERROR
Une spécification de frontière ne peut être respectée dans une variable structurée.

BIT SPECIFICATION OUT OF RANGE
Le numéro d'un bit est hors des limites de la chaîne de bits dont il fait partie.

Requête EXIT (Sous-environnement d'analyse)

Format : EXIT

Utilisation : Cette requête permet de rentrer dans l'environnement de définition pour pouvoir modifier les structures au niveau de leur description symbolique.

Messages : DEFINITION ENVIRONMENT ...ENTER REQUEST
Le système signale que l'environnement de définition est de nouveau accessible. Seules les requêtes de cet environnement sont alors valables.

Requête FIND (sous-environnement d'analyse)

Format : FIND { CORE < valeur > [< alignement >] | PATH < nom > [< adresse >] }

Utilisation : Cette requête permet de faire des recherches sur des valeurs contenues dans l'image mémoire examinée ou dans les contextes (chemins) mémorisés par des requêtes REMEMBER.

L'option CORE indique que la recherche se fait dans l'image mémoire examinée.

L'option PATH indique que la recherche se fait dans les contextes.

< valeur > est la valeur à rechercher et doit être spécifié selon les différents types de variables valides (entier, chaîne de caractères, etc...)

< alignement > indique si les adresses sur lesquelles se portent la recherche de la valeur doivent respecter un alignement (D double-mot, W mot, H demi-mot, ou B octet). Si aucun alignement est spécifié D est pris par défaut.

< nom > indique le nom du contexte que l'on veut rechercher.

Lorsqu'en plus <adresse> est spécifiée, une fois le contexte trouvé, une recherche sera faite pour déterminer si une table d'adresse <adresse> existe.

Messages : FOUND AT *****

La valeur recherchée a été trouvée à l'adresse *****

VALUE NOT FOUND

La valeur spécifiée n'est pas trouvée dans les limites de la recherche (§ requête SET LIMIT).

INVALID BOUNDARY

L'alignement spécifié n'est pas connu.

EMPTY STRING

La valeur chaîne de caractères est une chaîne vide ("")

INVALID ADDRESS

L'adresse spécifiée est hors des limites de recherche.

NO PATH KEPT

Aucun chemin n'a été mémorisé. Donc la recherche ne peut se faire.

NOTE : Les limites de la recherche d'une valeur donnée peuvent être modifiées par la requête SET.

Requête FORGET (sous-environnement d'analyse)

Format : FORGET < nom >

Utilisation : La requête FORGET est destinée à supprimer un chemin gardé précédemment par la requête REMEMBER.

< nom > est le nom du chemin à supprimer.

Messages : PATH NOT FOUND
Aucun chemin ne correspond au nom donné.

NO PATH KEPT
Aucun chemin n'a été mémorisé.

Requête MODIFY (Sous-environnement de définition)

Format : MODIFY < nom structure >

Utilisation : Cette requête utilise un mini-éditeur pour la correction de la variable structurée spécifiée.

Messages : SYMBOL NOT FOUND
Aucune variable ne correspond à l'identificateur spécifié.

USE SUPPRESS AND DEFINE REQUESTS
Ce message est imprimé lorsque le nom spécifié correspond à une variable simple. Pour la modifier il suffit de supprimer la définition correspondante et de rentrer une nouvelle définition.

Requête NEXT (Sous-environnement d'analyse)

Format : NEXT [< nombre > [USING <nom pointeur>]]

Utilisation : Lorsque la structure "qualifiée" est sous forme de liste, la requête NEXT permet de se déplacer de n tables en avant, à partir de la table courante. Si < nombre > est omis , il sera pris égal à 1. Le pointeur utilisé pour le déplacement est le premier rencontré dans la description symbolique. Au cas où il en existe plusieurs, le pointeur désiré peut être spécifié avec le mot-clé USING.

Messages : LAST TABLE REACHED
La dernière occurrence de la structure est atteinte avant d'avoir épuisé les occurrences spécifiées. La dernière occurrence est active .

NO QUALIFIED TABLE
Avant l'utilisation de la requête NEXT, aucune structure n'a été "qualifiée" par une requête QUALIFY.

INVALID NUMBER
La syntaxe du nombre donné en paramètre est incorrecte.

UNKNOWN POINTER NAME
Le nom du pointeur n'a pas été trouvé dans la description symbolique.

Requête QUALIFY (sous-environnement d'analyse)

Format : QUALIFY { < nom structure > [AT < adresse >] | < pointeur > → < pointeur > → ... → < nom structure > }

Utilisation : Pour accéder aux parties d'une variable structurée, l'utilisateur doit auparavant "qualifier" la structure au moyen de la requête QUALIFY.

< non structure > est le nom de la variable structurée.

AT < adresse > doit être utilisé pour spécifier la base absolue d'une variable structurée, dans le cas où dans la déclaration symbolique la base n'est pas donnée.

< pointeur > → < pointeur > → ... sont des adresses symboliques ou absolues des mots représentant une chaîne de pointeurs vers la variable structurée à "qualifier". Cette notation est à utiliser lorsque dans la déclaration symbolique la base de la variable structurée n'est pas spécifiée, ou encore lorsque la base utilisée ne correspond pas à celle donnée dans la déclaration.

Exemple :

Soit l'en-tête de la structure suivante :

UTABLE STRUC BASE(RUNUSER)

La requête QUALIFY UTABLE a pour effet d'activer la

structure UTABLE dont la première occurrence est pointée par RUNUSER.
Si la requête QUALIFY GPR(11) → UTABLE est émise, alors l'occurrence de UTABLE pointée par le registre 11 est activée.

Messages : NOT STRUCTURE TYPE
Le nom spécifié en paramètre n'est pas l'identificateur d'une variable structurée.

STRUCTURE NOT FOUND
Aucune déclaration de variable ne correspond au nom donné en paramètre.

NOT ENOUGH SPACE, MODIFY "OPTIONS" FILE.
La pile servant à l'activation des structures est saturée, car le nombre maximum de requêtes QUALIFY successives autorisées est atteint.

ALREADY QUALIFIED
La structure spécifiée en paramètre est la structure active
La requête est ineffective dans ce cas.

UNKNOWN BASE : xxx
La base xxx donnée dans la définition de la structure n'a pas été trouvée ou n'est pas accessible à partir de la structure active.

BOUNDARY ERROR IN POINTER OR OFFSET
L'adresse absolue ou relative d'un pointeur n'est pas un multiple de l'adresse mot.

VALUE ERROR IN POINTER
La valeur trouvée dans un pointeur est égale à zéro ou est supérieure à la taille mémoire de la machine réelle.

INCORRECT SUBSCRIT
Une variable indicée se trouvant dans la liste des pointeurs n'est pas correctement parenthésée.

Requête QUERY (Sous-environnements de définition et d'analyse)

Format : QUERY

- ACCESS
- ACTIVE
- ADDRESS [< nom variable >]
- CORE
- LIMIT
- OFFSET < nom > [IN <nom structure >]
- PATH [< nom >]
- TRACE

Utilisation : Cette requête donne divers renseignements sur le fonctionnement de XCP. Ces renseignements sont obtenus à l'aide des différents mots-clés donnés en paramètre.

ACTIVE donne le nom de la structure active avec la liste des adresses des occurrences activées (§ requête BACK et NEXT)

ACCESS donne les noms des structures accessibles à partir d'une structure active.

ADDRESS donne l'adresse absolue et relative (offset) d'une variable appartenant à la structure active. Si le nom est omis, l'adresse fournie est celle de la structure.

CORE donne l'origine de la mémoire de CP-67 à examiner. Elle peut provenir d'une bande, d'un disque, d'un fichier de "spooling" ou est la mémoire réelle du système CP-67.

LIMIT donne les adresses inférieure et supérieure utilisées dans la requête FIND. Ces adresses représentent le domaine d'action de FIND, pour la recherche des valeurs.

OFFSET donne l'adresse relative (offset) d'une variable appartenant à une structure. Au cas où plusieurs variables de même nom existent, le nom de la structure doit être spécifié après le mot-clé IN pour éviter toute ambiguité.

PAGE donne les adresses des différentes pages de l'image-mémoire ou de l'instantané de CP-67 auquel l'utilisateur a accès.

PATH donne la liste des noms de tous les chemins (contextes) mémorisés par des requêtes REMEMBER. Si < nom > est spécifié, le chemin ayant ce <nom> est listé, s'il existe.

TRACE donne la liste de toutes les structures "qualifiées" successivement par les requêtes QUALIFY, ainsi que celle des adresses des occurrences de ces structures.

Exemple

```
Qualify      READERS    SFBLOK
next 2
Qualify  MRDEBLOK
```

La requête QUERY TRACE a comme effet :

```
SFBLOK  WAS  QUALIFIED
#4B00   #4B30    #3AC0
MRDEBLOK  WAS  QUALIFIED
#5CD0
```

Messages : UNKNOWN KEYWORD

Le mot-clé spécifié ne correspond à aucun mot connu par la requête QUERY.

SYMBOL NOT FOUND

Les noms spécifiés avec les mots-clés ADDRESS OFFSET ou PATH ne sont pas des identificateurs de variables ou de chem connus.

NO AVAILABLE CORE

Aucun instantané n'a pu être lu par XCP. Ceci est dû soit au fait que l'utilisateur n'a pas spécifié correctement les paramètres d'utilisation de XCP, soit qu'il n'a pas émis la requête SNAP.

NO QUALIFIED STRUCTURE

Le mot-clé TRACE a été spécifié alors qu'aucune structure n'a été qualifiée.

Requête REMEMBER (Sous-environnement d'analyse)

Format : REMEMBER < nom >

Utilisation : Les structures "qualifiées" successivement avec leurs adresses absolues peuvent être gardées et activées ensemble ultérieurement.
Cette mémorisation se fait à l'aide de la requête REMEMBER.

< nom > est le nom donné au chemin (contexte) que l'on veut mémoriser.

NO QUALIFIED STRUCTURE
Aucun chemin ne peut être gardé, car il n'y a pas de structure "qualifiée".

NAME ALREADY EXISTS
Le nom donné correspond déjà à un chemin existant.

Requête SAVE (Sous-environnement d'analyse)

Format : SAVE { SYMBOLS | <nom structure> }

Utilisation : Les variables définies au moyen de la requête DEFINE ne sont rendues permanentes qu'à la demande de l'utilisateur, soit par la requête ENDEF soit par la requête SAVE.

SYMBOLS Ce paramètre est à spécifier lorsque les variables à sauvegarder sont des variables simples.

<nom structure> précise le nom de la structure. Celle-ci est alors réécrite entièrement dans un fichier de nom, le nom donné et de type STRUC.

Messages : SYMBOL NOT FOUND
Le symbole spécifié en paramètre ne correspond à aucune variable définie.

I/O ERROR WHILE WRITING
Une erreur d'entrée-sortie s'est produite lors de l'écriture de la définition de la variable sur disque. L'exécution de la requête est supprimée.

Remarque | Si une défaillance de CP-67 se produit au moment de l'exécutior de la requête SAVE, l'ancien fichier correspondant à l'ancienne définition de la variable a pour nom <<TEMP>> et pour type <<TYI

Requête SEARCH (Sous-environnement d'analyse)

Format : SEARCH <nom> <opérateur logique> <valeur> [USING <nom pointeur>]

Utilisation : Lorsque la structure "qualifiée" est sous forme de liste, la requête SEARCH peut être employée pour rechercher des valeurs satisfaisant certaines conditions.

<nom> est le nom de la variable simple sur laquelle porte la recherche. Elle doit obligatoirement appartenir à la structure "qualifiée".

<opérateur logique> est l'un des 6 opérateurs de comparaison =, ¬=, <, >, <= , >=

<valeur> est une valeur correspondant au type de la variable spécifiée.

<nom pointeur> L'accès aux éléments successifs de la liste est fait d'après les déclarations contenues dans la structure. Lorsqu'il y a plusieurs pointeurs possibles, le premier pointeur est pris. Un pointeur précis peut être également spécifié en utilisant le mot-clé USING.

Messages : NO QUALIFIED STRUCTURE
La requête SEARCH ne peut être utilisée car aucune structure n'est "qualifiée".

LAST TABLE REACHED

La dernière occurrence de la structure est atteinte et la condition n'est pas satisfaite.
La dernière occurrence est active.

CONDITION SATISFIED

La condition spécifiée est satisfaite et l'occurrence de la structure contenant la variable satisfaisante à la condition, est active.

INTERNAL TYPE ¬= REF

La variable spécifiée après le mot-clé USING n'est pas de type REF.

Requête SET (Sous-environnement de définition et d'analyse)

Format : SET { LIMIT [< adr1 > < adr2 >] | CORE { REAL | RPRT | TAPE | DISK } }

Utilisation : La requête SET sert à spécifier certains paramètres de fonctionnement de XCP.

LIMIT indique les limites de recherche d'une valeur donnée par la requête SEARCH.

<adr1> est l'adresse de début de la zone sur laquelle se fait la recherche.

<adr0> est l'adresse de fin de la zone.
Si aucune adresse n'est spécifiée, la recherche se fait sur toute la mémoire à examiner.

CORE spécifie la provenance de l'instantané à examiner.

REAL l'instantané sera pris avec la requête SNAP

RPRT l'instantané se trouve sur une unité spéciale de "spooling"

TAPE l'instantané se trouve sur bande magnétique d'adresse X'181'.

DISK l'instantané se trouve sur disque sous forme de fichier dont le nom et le type seront spécifiés ultérieurement (§ chapitre 3).

Remarque

Les spécifications sur l'origine de l'instantané peuvent être également données au moment de l'activation de XCP.

Messages : SOME PAGE(S) NOT AVAILABLE
Ce message est imprimé lorsque l'une des adresses spécifiées n'appartient pas à la mémoire examinée.

Requête SHOW (sous-environnement d'analyse)

Format : SHOW { < nom1 > ... < nomn > | < pointeur > → <nom 1> ... < nom 2 > }

Utilisation : La requête SHOW sert à demander l'impression des variables symboliques selon leur type déclaré. Les noms donnés en paramètres doivent appartenir à la structure courante "qualifiée" sinon ils doivent être "qualifiés" temporairement en utilisant les pointeurs.

<nom1> le nom des variables appartenant à la structure "qualifiée".

<pointeur> → etc..<nom> si le nom appartient à une structure non encore qualifiée, l'accès à cette variable doit être indiqué par des pointeurs. <nom> est dite alors variable "qualifiée temporairement".

Messages : XXX NON-EXISTENT
La variable de nom XXX n'a pas été trouvée.

POINTS TO XXX (UNDEFINED)
Dans une chaine de pointeurs, l'adresse référencée XXX par un pointeur n'a pas été trouvée.

NOT REFERENCE TYPE
Un pointeur utilisé n'est pas déclaré comme REF.

ERROR IN SUBSCRIPT
Une erreur est trouvée dans une variable indicée (erreur de parenthésages, valeur incorrecte, etc...)

Remarque

Lorsqu'une variable est un tableau et qu'aucun indice n'est spécifié, tous les éléments de ce tableau seront imprimés.

Requête SNAP (Sous-environnement d'analyse)

Format : SNAP { CORE / TAPE }

Utilisation : Cette requête permet de prendre un instantané de la mémoire de CP-67.

CORE spécifie que l'instantané doit être mis directement en mémoire virtuelle de la machine activant XCP.

TAPE spécifie que l'instantané doit être mis sur une bande magnétique d'adresse virtuelle x'181'

Messages : FATAL ERROR DURING SNAP
Une erreur a été détectée et réfléchie par CP-67 durant l'exécution de la requête.

TAPE 181 NOT ATTACHED, REPLY "GO" WHEN READY
Ce message prévient l'utilisateur qu'à l'adresse indiquée aucune bande n'est attachée. Si au bout de 3 tentatives, aucune bande n'est acquise, la requête SNAP est arrêtée.

Requête STOP (tout environnement)

Format : STOP

Utilisation : Cette requête est destinée à terminer une session de XCP et à redonner le contrôle au système CMS.

Message : Aucun.

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

1 J.R.ABRIAL, G.BEAUME, R.MORIN et G.VIGLIANO

Projet Socrate. Spécifications générales.
IMAG, Août 1970.

2 P.ALSBERG

Extensible data features in the operating system language OSL/2
Third Symposium on Operating Systems Principles.
Palo Alto, October 1971.

3 M.ADIBA

Editeurs par contexte pour systèmes conversationnels à partage de temps.
Thèse 3e Cycle. IMAG, Avril 1971.

4 M.BERTHAUD, D.CLAUZEL et C.JACOLIN

GSL Définition du langage.
Centre Scientifique IBM de Grenoble.

5 BAYER, GRIES, PAUL & WIEHLE

The Alcor Illinois 7090/7094 post-mortem dump.
Communications ACM.Vol 10 n°12 December 1967.

6 L.BOLLIET

Notation et processus de traduction des langages symboliques
Thèse d'Etat. IMAG,Juin 1967.

7 J.M.CAGNAT

Structure de représentation des données en ordinateur. Application aux traitements graphiques.
Thèse 3e Cycle, IMAG, Février 1971.

8 CORBATO, SALTZER

Multics, the first seven years.
Spring Joint Computer Conference 1972.

9 R.GRISHMAN

The debugging system AIDS.
Spring Joint Computer Conference 1970.

10 C.HANS et A.AUROUX

Notion de machines virtuelles.
Revue de l'AFCET n°15, Décembre 1968.

11 IBM System/360

Principles of operation.
A 22-6821

12 IBM System/360

Model 67. Functional characteristics.
A 27-2719.

13 IBM System/360

Component description. IBM 2250 display unit model 1.
A 27-2701.

14 IBM Control Program-67/Cambridge Monitor System

CP-67 Program Logic Manual
GY 20-0590

15 IBM Control Program-67/Cambridge Monitor System

CMS Program Logic Manual.
GY 20-0591

16 IBM Control Program-67/Cambridge Monitor System

Installation Guide.
GH 20-0857

17 IBM Control Program-67.Cambridge Monitor System

CMS User's Guide
GH 20-08591

18 IBM System/360. Operating System

Graphic programming services for IBM 2250 display unit.
C 27-6909

19 IBM PL/1 Subroutine Library

Program logic manual
Y28-6801

20 IBM System/360. System Reference Library

PL/1 Reference Manual
C28-8201

21 IBM System/360. Operating System

Assembler language
C28-6514

22 IBM System/360. Operating System

Supervisor and Data management services.
C28-6646

23 IBM System/360 TSS

Time Sharing Support System
GC 28-2006

24 IBM System/370

Principles of operation
GA 22-7000

25 W.JOSEPHS

An on-line machine language debugger for OS/360.
Fall Joint Computer Conference 1969.

26 D.KNUTH

The art of computer programming .
Volume 1.

27 N.LAURANCE

A compiler language for data structure.
23 th ACM National Conference.

28 J.P.LEHEIGET

Généralisation de la notion d'espace virtuel. Applications aux systèmes CP/CMS.
Thèse CNAM.CUEFA Grenoble Juillet 1972.

29 MILLS

Syntax-directed documentation for PL/360
Communications ACM Vol 13 n°4, April 1970.

30 NGUYEN THAN THI

A graphic conversational aid for errors searching and teaching of an operating system.
Workshop on virtual computer systems, March 26 1973
Harvard University, Cambridge.

31 P.POOLE

Debugging and testing. Advanced course on software engineering.
Munich February 1972.

32 B.RANDELL

Operating Systems : the problems of performance and reliability.
IFIP. Ljubljana, August 1971.

33 S.ROSEN

Programming systems and languages.

34 R.RUSTIN

Debugging techniques in large system.

35 SCHWEMM

Experienced gained in the development and use of TSS.
Spring Joint Computer Conference 1972.

36 R.WILLIAMS

A survey of data structures for computer graphic systems.
Computer Surveys. Vol 3 n°1, March 1971.

37 N.WIRTH

A programming language for the 360 computers
Stanford University.

38 W.WULF, D.RUSSEL & A.HABERMANN

BLISS : A language for systems programming.
Communications ACM Vol 14 n°12 December 1971.

39 L.ZIMMERMAN

GBUG : a graphic aid to on-line program debugging
IBM Corporation 360.D.4.0.001

40 L.ZIMMERMAN

On line program debugging. A graphic approach.
Computers and automation. November, 1967.

41 A.VAN DAM & D.E.RICE

On-line text editing: A survey
Computing Surveys
Vol 3 n°3, September, 1971.

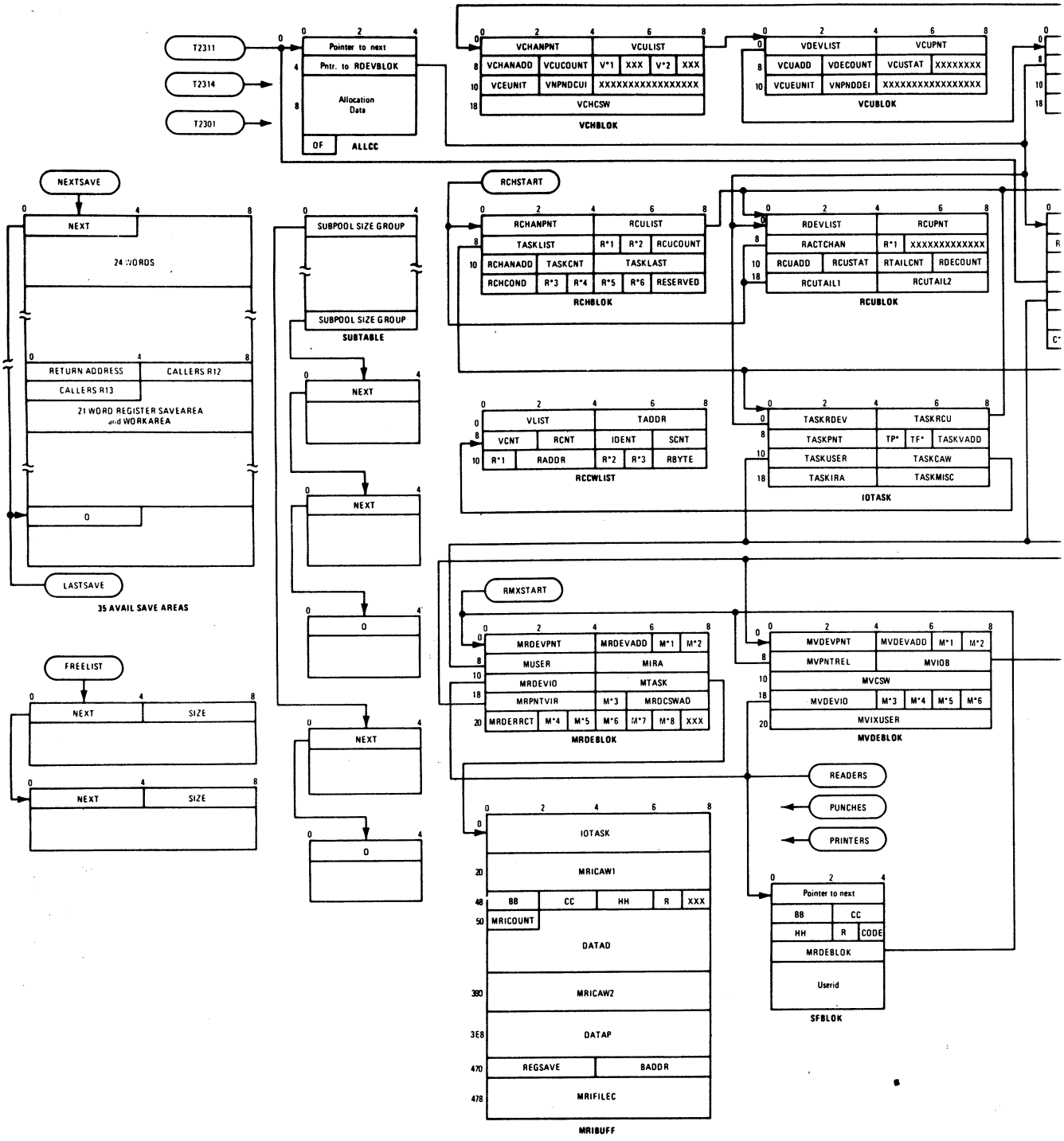
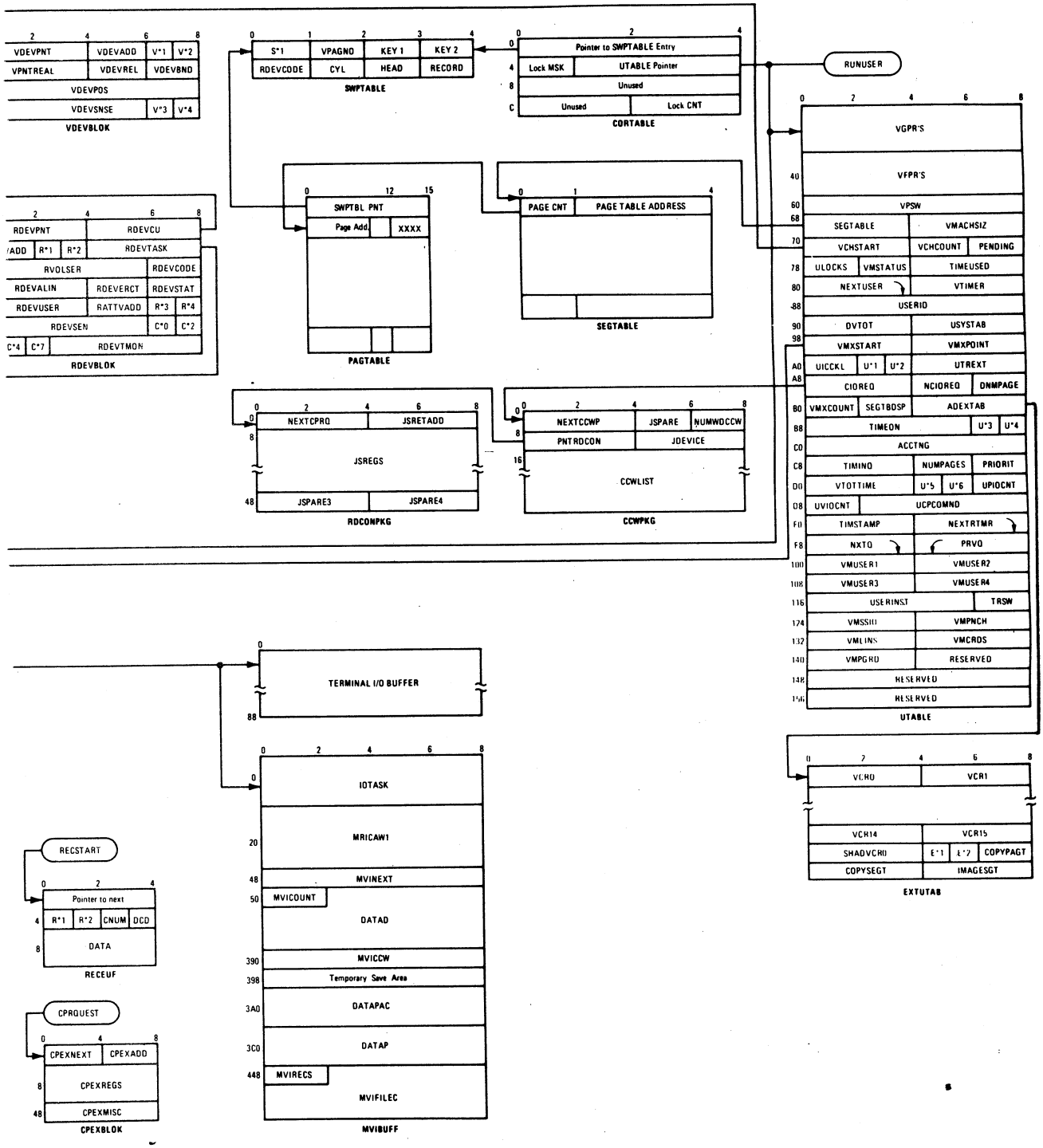


Figure 3.1.
Blocs de contrôle


VDEVPNT
VDEVADD
V*1
V*2
VPNTREAL
VDEVREL
VDEVBND
VDEVPOS
VDEVSNSE
V*3
V*4
VDEVBLOK
S*1
VPAGNO
KEY 1
KEY 2
RDEVCODE
CYL
HEAD
RECORD
SWPTABLE
Pointer to SWPTABLE Entry
Lock MSK
UTABLE Pointer
Unused
Unused
Lock CNT
CORTABLE
RUNUSER
RDEVPNT
RDEVCU
RDEVTASK
RVOLSER
RDEVCODE
RDEVALIN
RDEVERCT
RDEVSTAT
RDEVUSER
RATTVADD
R*3
R*4
RDEVSEN
C*0
C*2
C*4
C*7
RDEVTMON
RDEVBLOK
SWPTBL PNT
Page Add.
XXXX
PAGTABLE
PAGE CNT
PAGE TABLE ADDRESS
SEGTABLE
VGPR'S
VFPR'S
VPSW
SEGTABLE
VMACHSIZ
VCHSTART
VCHCOUNT
PENDING
ULOCKS
VMSTATUS
TIMEUSED
NEXTUSER
VTIMER
USERID
DVTOT
USYSTAB
VMXSTART
VMXPOINT
UICCKL
U*1
U*2
UTREXT
CIOREQ
NCIOREQ
DNMPAGE
VMXCOUNT
SEGTBDSP
ADEXTAB
TIMEON
U*3
U*4
ACCTNG
TIMING
NUMPAGES
PRIORIT
VTOTTIME
U*5
U*6
UPIOCNT
UVIOCNT
UCPCOMND
TIMSTAMP
NEXTRTMR
NXTQ
PRVQ
VMUSER1
VMUSER2
VMUSER3
VMUSER4
USERINST
TRSW
VMSSID
VMPNCH
VMLINS
VMCRDS
VMPGRD
RESERVED
RESERVED
RESERVED
UTABLE
NEXTCPRQ
JSRETADD
JSREGS
JSPARE3
JSPARE4
RDCONPKG
NEXTCCWP
JSPARE
NUMWDCCW
PNTRDCON
JDEVICE
CCWLIST
CCWPKG
TERMINAL I/O BUFFER
IOTASK
MRICAW1
MVINEXT
MVICOUNT
DATAD
MVICCW
Temporary Save Area
DATAPAC
DATAP
MVIRECS
MVIFILEC
MVIBUFF
VCR0
VCR1
VCR14
VCR15
SHADVCR0
E*1
E*2
COPYPAGT
COPYSEGT
IMAGESGT
EXTUTAB
RECSTART
Pointer to next
R*1
R*2
CNUM
DCD
DATA
RECEUF
CPRQUEST
CPEXNEXT
CPEXADD
CPEXREGS
CPEXMISC
CPEXBLOK

Requête LIST (Tout environnement)

Format : LIST { < nom > < type > [IN < nom >] / ALL }

Utilisation : Cette requête donne la liste des variables ou des macro-requêtes définies au cours de la session ou antérieurement.

<nom> Le premier paramètre spécifie le nom de la variable ou
ALL de la macro. Si le mot-clé ALL est présent toutes les définitions sont listées.

<type> spécifie le type valide d'une variable ou MAC (macro-requête).

IN <nom> Le nom donné ici est le nom d'une variable structurée. la recherche de la variable ne se fait que dans le cadre de cette structure.

Messages : UNKNOWN KEYWORD
Les mots-clés spécifiés ne correspondent ni à ALL ni à IN

NOT STRUCTURE TYPE
Le nom spécifié après le mot-clé IN ne correspond pas à un identificateur d'une structure.

ERROR WHILE READING FILE
Une erreur a été détectée lors de la lecture du fichier contenant une macro-requête.
L'exécution de la macro-requête est alors supprimée.

MACRO DOES NOT EXIST
Aucune macro-requête ne correspond au nom spécifié.