La mise au point de système de programmation grâce à un couplage de machines virtuelles

Patrick Lefèvre

n° d'ordre

THESE

présentée à

l'Université Scientifique et Medicale de Grenoble

pour obtenir

le grade de Docteur de Troisième Cycle Informatique

par

Patrick LEFEBVRE

SPY : UN SYSTEME DE CONTROLE

LA MISE AU POINT DE SYSTEMES DE PROGRAMMATION GRACE A UN COUPLAGE DE MACHINES VIRTUELLES

Thèse soutenue le 5 juillet 1972 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
Mle	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
Mle 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.

Je tiens à remercier :

Monsieur le Professeur Noël GASTINEL, Directeur du Laboratoire de Calcul qui m'a fait l'honneur de présider le Jury de cette thèse et qui a accordé un vif intérêt à mon travail.

Monsieur Louis BOLLIET, Professeur à l'Institut Universitaire de Technologie de Grenoble qui m'a orienté vers la recherche en programmation et m'a encouragé dans mes travaux.

Monsieur Michaël GRIFFITHS, Maître de Conférence à l'Université Scientifique et Médicale de Grenoble, qui s'est intéressé avec sympathie à mon travail et a aimablement accepté de faire partie du Jury.

Monsieur Claude HANS, Ingénieur au Centre Scientifique IBM-France qui est à l'origine du sujet de cette thèse et n'a ménagé ni son temps ni ses conseils précieux pour me diriger et m'encourager dans mes travaux.

Tous mes collègues du Laboratoire et du Centre Scientifique, les membres de l'équipe de maintenance des systèmes CP-CMS qui m'ont aidé dans la réalisation de ma tâche grâce à leurs remarques, leurs conseils ou contribution active, en particulier Messieurs AUROUX, BELLOT, DUPUY, GATEAU, LE HEIGET, PERRONEAU.

Je tiens à remercier les services de dactylographie et de tirage à qui je dois la réalisation de cet ouvrage.

TABLE DES MATIERES

CHAPITRE - II - DESCRIPTION D'UN SYSTEME DE CONTROLE : SPY

CHAPITRE III - DESCRIPTION LOGIQUE DE SPY

CHAPITRE - IV - MANUEL DE L'UTILISATEUR DU SYSTEME SPY

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

La phase finale de la réalisation d'un système de programmation, la mise au point de l'ensemble, est surement l'activité la plus éprouvante pour la personne -ou les personnes- chargées de ce travail. Cela est dû essentiellement au fait qu'il est difficile de définir tous les cheminements dans la masse des programmes mis en commun, de prévoir toutes les combinaisons de paramètres et de créer les conditions exactes de fonctionnement.

Les fonctions de test permettent uniquement de découvrir des erreurs de programmation, mais ne permettent pas de déceler l'absence de ces fautes. De plus le responsable de ce travail n'a pas à sa disposition des aides à la mise au point en rapport avec la complexité du problème à résoudre. Généralement la mauvaise situation à détruire n'est matérialisée que par une image de mémoire du calculateur prise au moment où le système de programmation refuse obstinément de travailler. Il peut aussi utiliser les fonctions cablées disponibles sur le pupitre de la machine.

La programmation met en évidence un fait paradoxal : plus les algorithmes deviennent compliqués, plus les programmes sont énormes et moins le programmeur peut recevoir de l'aide de l'ordinateur.

Celui qui est chargé d'un système de programmation est toujours fataliste, il affirmera : "Tous les programmes importants ont toujours des erreurs quelque soit leur âge".

Ces erreurs peuvent être classées en deux catégories :

- Les erreurs logiques :

Le système de programmation fonctionne très mal, ou pas du tout. C'est-à-dire qu'il donne des réponses erronnées, qu'il a des réactions non prévues ou plus simplement il s'arrête incapable de poursuivre un quelconque travail. En général ces erreurs sont provoquées par une conjoncture exceptionnelle et elles apparaissent de façon aléatoire. Par là même on peut difficilement les reproduire car elles sont provoquées la plupart du temps, par un événement très antérieur à leur matérialisation.

- Les erreurs de performance :

Apparemment le système de programmation fonctionne normalement, mais il gère ses ressources de façon inefficace. Par exemple dans le cas d'un système conversationnel le temps de réponse est exagérément long ou bien le temps passé à exécuter des tâches du superviseur est prohibitif. Elles sont provoquées généralement par une mauvaise utilisation des ressources physiques.

Pour définir un compromis acceptable entre les erreurs restantes et les travaux exécutés un patient travail de mise au point est nécessaire.

En général, la construction d'un système est menée à bien en quatre phases. Tout d'abord une phase de réflexion permettant de définir les buts et les moyens.

Ensuite la phase de construction proprement dite : on choisit les algorithmes généraux du système. A ce niveau prend très souvent place la réalisation d'un modèle simulant le système. On écrit les programmes correspondant aux algorithmes élémentaires qui représentent les différentes fonctions.

Une première partie de la mise au point peut être effectuée maintenant. Les erreurs grossières de ces éléments sont détectées et corrigées. Chaque fonction du système est testée séparément. Un arsenal non négligeable d'aides sont accessibles au programmeur qui peut franchir assez facilement cette étape.

En dernier, il faut assembler tous ces éléments ; dès lors il faut se résoudre à ne disposer que d'un petit nombre d'outils de mise au point élémentaires bien que le plus souvent, un certain nombre d'outils programmés aient été incorporés au système pour apporter quelques facilités.

On peut se demander pourquoi cet état de fait existe, pourquoi le programmeur ne dispose en général que de fonctions "électroniques" de mise au point -voyants lumineux du pupitre de sa machine, boutons et clés- et de quelques aides programmées simulant ces fonctions.

La mise en évidence des deux types d'erreurs que nous avons citées précédemment ne peut se faire que si le système de programmation fonctionne réellement. C'est-à-dire, par exemple, il contrôle des travaux réels dans un environnement qui est celui dans lequel il réside normalement. Ce système ne doit pas fonctionner dans un environnement privilégié de tests, qui peut affecter les caractéristiques du système, qui peut modifier l'enchaînement des algorithmes, et qui peut introduire de nouveaux paramètres. Les programmes composant le système en cours de tests sont influencés par de très nombreux paramètres extérieurs dont la principale caractéristique est une haute fréquence de modification et d'apparition. En général, l'état instantané d'un ensemble de conditions n'est pas prévisible et il est difficile de le reproduire, voire même impossible.

Pour aider le programmeur à découvrir ses erreurs, il faut lui offrir des moyens de détection de la condition anormale alors qu'elle est à l'état embryonnaire. Par exemple on désire trouver le programme qui dans certains cas particuliers détruit une zone mémoire réservée à une autre fonction. Or il ne faut pas modifier l'environnement du système pour découvrir cette erreur qui dans de nouvelles conditions serait susceptible de ne plus apparaître. Mais aussi, pour découvrir cette erreur, les fonctions de mise au point doivent examiner continuellement la zone de mémoire qui est susceptible d'être détruite afin de connaître l'instant exact de sa modification. Pour ce faire il faut faire appel à une fonction compliquée dont l'exécution modifiera sûrement certains paramètres du système.

Nous nous trouvons confronter à une double contradiction : ne pas affecter le système de programmation impose la présence d'outils invisibles, peu encombrants et qui ne soient activés que très rarement. Ces outils doivent être construits de façon totalement indépendante du système. C'est-à-dire qu'ils ne doivent mettre en jeu aucune des fonctions du système testé. S'il en était autrement une erreur pourrait rendre inutilisable les outils de mise au point.

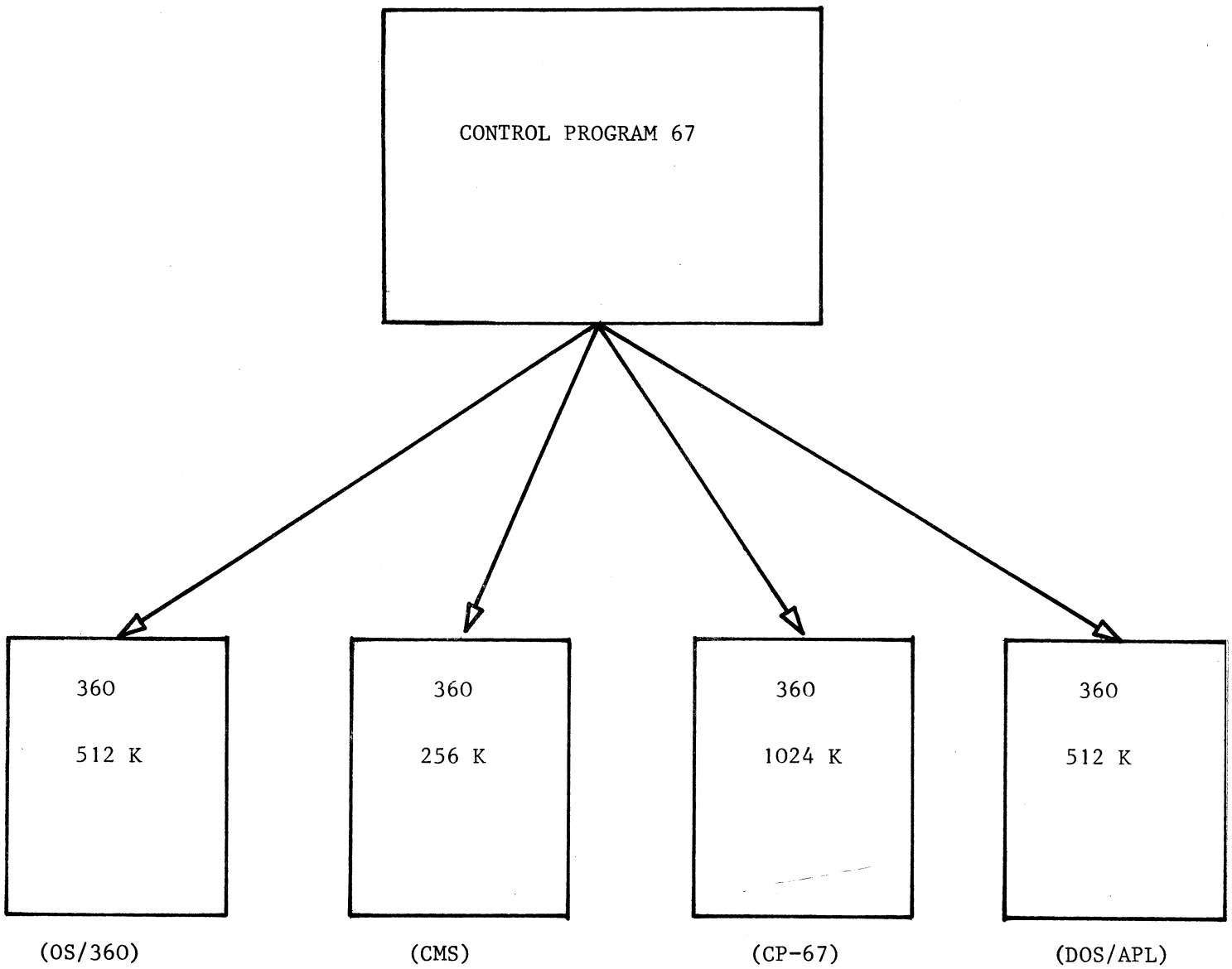
Mais le caractère aléatoire, fugitif et non immédiatement fatal des conditions anormales que nous recherchons impose un contrôle pratiquement constant, des outils de mise au point programmés puissants, donc en général encombrants et volumineux. Leur activation provoquera des perturbations plus au point profondes dans le système. Ils ne doivent pas utiliser des composants du système testé. De par leur taille, de par leur complexité, il est difficile de les protéger contre les erreurs de leur "cobaye".

Nous nous proposons d'apporter une solution qui résorbe cet antagonisme. Nous avons défini et réalisé dans le système de programmation CP-67 un système de contrôle SPY qui permet la mise au point d'un système quelconque résidant sur une machine virtuelle. Ce système fournit à l'utilisateur des outils puissants mais qui ne modifient pas l'environnement du système testé.

Le système "CONTROL PROGRAM 67" (CP-67) développé au Centre Scientifique IBM de Cambridge aux Etats-Unis est un des rares systèmes qui met en oeuvre le concept de "machines virtuelles". Cette théorie partage les ressources disponibles dans un ordinateur (unité centrale, mémoire à accès aléatoire, unités d'entrée-sortie) de façon à ce que chaque utilisateur ait l'illusion de disposer d'un ordinateur complet pour lui seul. En conséquence, chaque utilisateur peut disposer de la configuration machine qui lui convient le mieux, peut activer le système de programmation de son choix sans réduire pour autant les possibilités de choix des autres utilisateurs (voir Figure I).

La réalisation de ce système de programmation peut se diviser en deux phases distinctes : d'une part générer et faire fonctionner une machine virtuelle, d'autre part gérer l'ensemble réel par un partage statique et dynamique entre plusieurs machines virtuelles (référence 14, 15, 12, 24).

On appelle machine virtuelle une configuration, prédéfinie au moyen d'une suite d'éléments, d'un calculateur complet, c'est-à-dire que les composants de la machine virtuelle sont représentés par un ensemble de données structurées décrivant l'unité centrale, la mémoire rapide, les dispositifs d'entrée-sortie. Les programmes nécessaires à la matérialisation du concept simulent le fonctionnement des dispositifs technologiques qui donnent à chaque utilisateur la possibilité de charger un système de programmation particulier et de l'utiliser.



Machines virtuelles générées par CP-67. Fig. I

CP-67, système générateur de machines virtuelles ; cette propriété très importante, justifie à elle seule la réalisation d'un tel ensemble de programmation. Une deuxième fonction fondamentale de ce système est de permettre le partage des ressources réelles entre toutes les machines virtuelles, ce qui implique deux autres nécessités :

* Eviter tout conflit d'une part entre les machines virtuelles et, d'autre part entre le système CP-67 lui-même et les machines virtuelles.

* Permettre des échanges d'informations entre les machines virtuelles.

De part la liberté laissée à chaque utilisateur d'une machine virtuelle, il est très facile de développer un système de programmation dans ce cadre. En effet, il dispose d'une configuration machine qu'il a lui-même défini, il peut exécuter des instructions privilégiées (instructions que l'on ne peut exécuter qu'en mode "maître"), il peut traiter ses propres interruptions, etc... Il y a donc un intérêt tout particulier à développer dans ce système CP-67 des outils de test permettant la mise au point des systèmes de programmation résidant sur les machines virtuelles.

Une première solution est d'utiliser soit les ensembles de mise au point généralement incorporés dans un système de programmation (environnement conversationnel DEBUG dans le système CMS par exemple), soit de construire un support de mise au point qui puisse s'adapter à tout système que l'on veut tester (support de mise au point PILOTE par exemple). Avec ce type d'ensemble il est difficile de ne pas modifier l'environnement des programmes en cours de mise au point. De même se pose aussi le problème de la protection des outils contre les erreurs et celui du contrôle de l'exécution du système testé.

Il est évident que ces trois problèmes, qui doivent être résolus dans le cadre de la mise au point, sont de même nature que ceux qui se posent aux constructeurs de système. Il est donc tentant de profiter des solutions apportées par le système CP-67, et d'incorporer, à côté de la fonction génération de machines virtuelles et de celle de multiprogrammation, une fonction de mise au point. Néanmoins, une telle réalisation manque de souplesse car elle entraîne des modifications profondes au superviseur et elle rend difficile la construction d'extension.

Nous avons donc adopté une solution médiane. D'une part nous avons voulu conserver les facilités de réalisation en implantant les outils de mise au point sur une machine virtuelle. D'autre part, nous avons voulu utiliser les mécanismes de protection du système CP-67 (deux machines virtuelles différentes sont totalement indépendantes), c'est pourquoi nous avons installé l'ensemble de mise au point sur une machine virtuelle et le système en cours de test sur une autre. Dès lors il faut concevoir une nouvelle relation entre machines virtuelles. Celle-ci doit permettre aux outils de mise au point d'examiner le système testé et la machine où il réside et elle doit leur permettre de contrôler l'exécution des programmes à mettre au point. Cette relation, que nous appelons relation d'inclusion, établit une hiérarchie entre machines virtuelles qui permet à un calculateur de "contrôler" le fonctionnement d'autre autre.

Nous voulons d'écrire les objectifs, les principes et la réalisation du système de mise au point SPY développé dans le système de programmation CP-67.

SPY a été conçu comme une suite logique des travaux de B. PETEUL avec l'objectif de fournir à un programmeur système un ensemble d'aides à la mise au point puissants, extensibles et utilisables quelque soit le système de programmation testé. Aucune supposition sur ce dernier n'est faite si ce n'est qu'il soit prévu pour un ordinateur de la série IBM 360.

Les principes de base de SPY sont la non modification de l'environnement des programmes testés et la protection totale des outils de mise au point. Nous avons conservé intact le cadre créé par le système CP-67 qui permet à un système de programmation de fonctionner dans des conditions très proches de la réalité. Les programmes en cours de test doivent totalement ignorer le contrôle auquel ils sont soumis, mais le programmeur doit disposer d'outils très puissants imposés par la difficulté des problèmes à résoudre. Le système de mise au point est installé sur une machine virtuelle, et le système de programmation à tester sur une autre, des liens privilégiés entre ces deux calculateurs sont développés. Ces liens autorisent l'examen du système "cobaye" et le contrôle de son exécution et assurent la protection de SPY. Une telle démarche intercale entre le système en cours de test et les composants électroniques un ensemble de fonctions totalement invisibles par le système.

Puisque nous disposons d'une machine virtuelle pour développer les outils de mise au point et les installer, nous pouvons définir un jeu d'aides fondamentaux et un mécanismes d'extension de cet ensemble de base.

SPY, système de mise au point conversationnelle autorise un contrôle continu de l'exécution et une observation instantannée des phénomènes qui se produisent. Nous avons donc développé des outils permettant à la fois un contrôle souple et intermittent et un contrôle rigoureux et minutieux.

L'élément de base de SPY est un simulateur d'instructions. Il fournit les moyens pour une exécution pas à pas, un arrêt sur référence à une zone mémoire donnée, une construction de la trace de l'exécution, ou d'un historique limité etc...

Néanmoins, cette méthode de travail très coûteuse n'est ni nécessaire ni souhaitable à tout instant. A cette exécution simulée a été couplée une exécution "presque normale" contrôlée par des points d'arrêts. Ceux-ci sont de deux sortes :

- Soient des points d'arrêt classiques implantés dans les programmes à exécuter par l'utilisateur.

- Soient des points d'arrêt sur des conditions prédéfinies telles que l'apparition d'un certain type d'interruption asynchrone, ou l'exécution d'une instruction privilégiée. Ces événements ont été choisis en raison du caractère particulier des programmes testés (seul un superviseur peut exécuter les instructions privilégiées manipulant les composants de la machine et généralement la politique de travail de ce type de programme est très influencée par le traitement des interruptions).

Cette deuxième méthode de travail, plus optimale, ne ralentit pratiquement pas l'exécution dans la mesure où le programmeur peut choisir les conditions particulières qui l'intéressent.

SPY utilise pour communiquer avec le monde extérieur des unités d'entrée-sortie qui lui sont personnelles (celles de sa machine virtuelle). Ceci permet de ne pas perturber le fonctionnement des dispositifs gérés par le système testé.

Un tel cadre peut se prêter à deux extensions de ce système de base :

- On peut fournir au programmeur des moyens très puissants d'extension des outils de mise au point. Il doit pouvoir en créer de nouveaux totalement différents de ceux existants et qu'il doit pouvoir combiner aux outils de base. Celà lui donne la possibilité de s'adapter à toute situation particulière.

- On peut développer des outils de mesures de performances afin de pouvoir répondre à des questions telles que :

  * combien de fois un composant particulier du système est utilisé.
  * combien d'instructions sont exécutées pour rendre un service particulier à un utilisateur.

Nous pensons que l'ensemble de ces qualités font de SPY un premier jalon dans la création d'outils de mise au point pour les systèmes de programmation et que les nouvelles propriétés des machines virtuelles permettront d'autres développements.

Dans le chapitres suivants nous décrirons les diverses possibilités de mise au point, utilisables par un programmeur, dans le système CP-67. De cette description nous pourrons extraire la définition d'un système de contrôle, d'une machine ESPION. Les principes généraux d'un système tel que SPY sont exposés et une réalisation pratique est décrite tant du point de vue fonctionnel que du point de vue logique. C'est la description du système de contrôle SPY lui-même.

CHAPITRE - I

LA MISE AU POINT DANS LE CADRE DU SYSTEME CP-67

Aux "temps héroïques" de la programmation une seule personne à la fois disposait du calculateur. Pour mettre au point son travail, elle utilisait uniquement des fonctions "hardware" : les boutons et les clés du pupitre de la machine. Cette solution était à peu près acceptable. En effet l'ordinateur ne possède aucun support de programmation, ce qui oblige l'utilisateur à s'occuper lui-même des diverses manipulations pour obtenir des résultats. Il en résulte une parfaite connaissance de la machine par le programmeur.

Au fur et à mesure que la programmation s'est éloignée des composants électroniques grâce en particulier aux langages de programmation, lorsque les ordinateurs sont devenus de véritables monstres, une telle pratique de mise au point a été abandonnée. Le temps d'immobilisation de la machine devenait prohibitif, et l'on ne pouvait employer qu'un nombre limité et non extensible d'outils de mise au point.

Depuis une dizaine d'années, et surtout depuis le développement des systèmes conversationnel on voit proliférer des aides à la mise au point en tant que composant d'un système de programmation particulier. Chaque langage de programmation, chaque système a le sien et si leur aspect extérieur diffère, les principes de réalisation restent les mêmes (référence I,26). Tous présentent une ou plusieurs contraintes réduisant leur efficacité dans le cas le plus compliqué : la mise au point d'un autre système, ou plus simplement, la mise au point du système auquel ils appartiennent.

En prenant comme exemple les ensembles de mise au point disponibles pour un utilisateur du système CP-67, nous expliciterons ces contraintes.

Lorsqu'un utilisateur, depuis un terminal, se fait reconnaître par CP-67, ce dernier génère la machine virtuelle qui a été attribuée à cette personne. Dès lors cette dernière, peut choisir, à quelques restrictions près, n'importe lequel des systèmes de programmation existants ou en cours de construction (évidemment prévu pour un ordinateur de la série 360) et l'activer sur sa machine virtuelle. L'utilisateur dispose de son système comme il l'entend, et en particulier, il peut tester et mettre au point une version expérimentale.

I-1 - SIMULATION DES FONCTIONS DE TESTS AU PUPITRE

Le système CP-67 donne l'illusion à chaque utilisateur, de disposer d'un calculateur pour lui seul. Il simule pour chacun une unité centrale, une mémoire rapide, des unités d'entrée-sortie etc..., mais aussi, il donne à l'utilisateur la possibilité de manipuler le pupitre de sa machine virtuelle. Ceci est réalisé grâce à une série de fonctions, appelées "fonctions consoles", utilisable depuis le terminal, qui représente alors le pupitre de la machine.

Quelques unes d'entre elles autorisent ce que nous avons appelé la mise au point pupitre. Au lieu de travailler avec les boutons et les clés du pupitre, il a, à sa disposition, six fonctions consoles.

Lorsque l'on veut procéder de la sorte, il faut tout d'abord arrêter le fonctionnement du calculateur virtuel afin d'examiner un certain nombre de mots mémoire privilégiés. Pour ce faire on peut procéder de deux manières différentes :

* soit par un arrêt aléatoire obtenu en appuyant la touche ATTN du terminal (c'est la simulation du bouton ARRET du pupitre). CP désactive alors la machine virtuelle -l'état de cette dernière est figé-, et le terminal devient le pupitre de l'ordinateur.

* soit par un arrêt à une adresse prédéfinie. La fonction SET ADSTOP XXXXXX (voir note) permet de fixer cette adresse. Ultérieurement, l'exécution de

---

Note : On trouvera la description exacte de ces fonctions dans la brochure "CP-67/CMS USER'S GUIDE" à partir de la page 492 (voir référence 14).

l'instruction se trouvant à cet emplacement, provoquera l'arrêt du calculateur. Le résultat final est identique à celui provoqué par la touche ATTN. L'utilisateur peut ainsi figer l'état de sa machine lorsque l'exécution de son programme atteintun stade critique.

Dans les deux cas, la machine virtuelle est bloquée et son état est figé ; l'utilisateur peut examiner et éventuellement modifier le contenu de la machine à l'aide des fonctions DISPLAY et STORE (c'est la simulation des boutons de même nom du pupitre).

Plus précisément, il est possible de visualiser et de changer le contenu de la mémoire rapide, des registres généraux et flottants, des registres de contrôle, du mot d'état programme etc...

A la suite de ce travail statique, la machine virtuelle peut être réactivée -l'exécution des programmes qui y sont implantés est à nouveau reprise, par exemple- grâce à la fonction BEGIN. C'est la simulation du bouton START.

Une dernière fonction qui n'est pas disponible sur une machine réelle, permet d'activer une trace du déroulement du programme en cours d'exécution sur le calculateur virtuel. Toutes les quantités, référencées ou modifiées par chaque instruction, sont communiquées à l'utilisateur qui peut, grâce à un jeu de paramètres, en sélectionner un sous-ensemble. Il peut aussi obtenir la liste des interruptions asynchrones qui ont été réfléchies à sa machine virtuelle.

D'après les principes mêmes du système CP-67, il est évident que l'utilisation de ces fonctions provoque uniquement l'arrêt de la machine virtuelle concernée.

De part ce mécanisme, on dispose d'un premier jeu d'outils pour la mise au point d'un système, jeu beaucoup moins coûteux d'emploi que son homologue réel. De plus, ces outils peuvent, éventuellement, être combinés avec les aides à la mise au point disponibles dans le système de programmation implanté sur la machine virtuelle. Examinons un peu plus en détail ceux du système CAMBRIDGE MONITOR SYSTEM (CMS) système conversationnel qui peut résider dans un calculateur virtuel généré par CP-67.

I-2 - ENSEMBLES DE MISE AU POINT DISPONIBLES DANS LE SYSTEME CMS

Le système CMS est un système conversationnel dialoguant avec un seul utilisateur. Il met à la disposition de ce dernier un langage simple de commandes pour utiliser en particulier :

* un éditeur de fichiers
* un programme de mise en page de texte
* un support de mise au point (environnement appelé DEBUG)
* les langages de programmation les plus usuels (FORTRAN, ALGOL, PL1, COBOL, ASSEMBLEUR 360 etc...).

Il donne la possibilité de charger et de faire exécuter des programmes appartenant à l'utilisateur.

Ce système conçu pour fonctionner aussi bien sur une machine virtuelle que sur une machine réelle est très simple puisqu'il ne s'occupe ni de multi-programmation ni de simultanéité, laissant ce soin à CP-67 (voir référence 14,15).

Contraitement à ce qui est habituellement autorisé dans un système classique, l'utilisateur à la possibilité d'exécuter ses programmes en "mode superviseur". Ceci rend particulièrement intéressante l'utilisation de CMS au cours du développement des divers composants d'un nouveau système.

Pour ce qui nous intéresse, il contient des aides à la mise au point que l'on peut ranger en deux classes :

* d'une part des outils de mise au point par des programmes écrits dans un langage de programmation évolué. En particulier on peut citer FORTBUG, accompagnateur pour programmes FORTRAN (voir référence 18).

* D'autre part un support de mise au point conversationnelle : PILOTE. Son but est de mettre à la disposition d'un utilisateur des outils qui permettent une mise au point facile et rapide de composants destinés à être ultérieurement intégrés dans un système de programmation.

C'est-à-dire que PILOTE est plus spécialement adapté aux programmes écrits en langage d'assemblage et il fournit des fonctions dont la puissance ne se justifie que pour des programmes représentant des algorithmes complexes (référence 26).

I-2-1 - OUTILS DE MISE AU POINT DISPONIBLES DANS PILOTE

PILOTE fournit deux types d'aides à la mise au point :

* d'une part des outils qui sont utilisés pendant l'arrêt du programme ou à son terme -outils statiques-.

* d'autre part des outils qui fonctionnent en parallèle avec le déroulement des instructions -outils dynamiques-. Cette dernière option a été développée à l'aide d'un simulateur d'instructions.

Pour les outils statiques, il est facile de les concevoir et de les fabriquer. La rencontre d'un point d'arrêt qui peut être placé là volontairement ou involontairement provoque une interruption asynchrone. L'exécution du programme testé est suspendue et l'environnement de mise au point prend le contrôle. Celui-ci doit sauvegarder l'état du programme en cours de test : registres généraux et flottants, compteur ordinal, etc... ; lire au terminal les commandes de mise au point qui lui sont demandées, puis exécuter les opérations correspondantes. Lorsque l'utilisateur demande la reprise de l'exécution de son programme, une restauration du contexte et une réactivation sont effectuées par l'environnement de mise au point.

Au contraire, les outils dynamiques doivent être actifs au cours du déroulement de chaque instruction. Essentiellement ils consistent en des arrêts conditionnels. Lorsque le programme testé remplit certaines conditions pré-définies, son exécution est suspendue, le contrôle est donné à l'environnement de mise au point et les outils statiques peuvent être utilisés. La recherche des conditions mentionnées ci-dessus est rendue possible par un interpréteur. C'est un programme qui analyse les instructions et qui produit un résultat identique à celui obtenu par une exécution normale. Après la phase d'analyse, l'interpréteur connaît toutes les caractéristiques de l'instruction courante : adresse, code opération,

opérandes, mémoires référencées, etc... et peut déterminer si les conditions sont vérifiées. Il est possible de construire des aides à la mise au point qui arrêtent l'exécution du programme testé pour certaines valeurs particulières, pré-définies par l'utilisateur, des caractéristiques. Le développement de cette classe d'outils est beaucoup plus difficile et complexe.

I-2-1-1 - Outils statiques de PILOTE

Ils sont disponibles quand l'exécution du programme testé est suspendue. Ce dernier se trouve alors dans un certain état caractérisé par la valeur du compteur ordinal, du contenu des registres et de la mémoire. Les outils de mise au point statique permettent d'examiner et de modifier cet état puis de réactiver le programme (référence 22, 26).

PILOTE dispose pour ces opérations, des requêtes suivantes :

X, GPR, CSW, CAW, PSW	pour examiner la mémoire, les registres, le mot d'état programme.
*X, GPR	pour examiner des emplacements mémoires dont l'adresse est le contenu soit d'un mot mémoire, soit d'un registre général.
STORE, SET	pour modifier la mémoire, les registres, le mot d'état programme.
GO	pour relancer l'exécution.

Citons enfin que les images partielles ou complètes de la mémoire sont accessibles grâce à la requête DUMP.

I-2-1-2 - Outils dynamiques de PILOTE

Nous sommes dans le cas où l'exécution des instructions du programme testé est entièrement simulée. Les caractéristiques obtenues lors de la phase d'analyse permettent de concevoir et de développer les outils suivants :

* les points d'arrêts sur condition. Les conditions actuellement acceptées sont au nombre de deux : référence à une certaine zone de la mémoire principale, exécution d'un certain type d'instruction.

* la trace des instructions exécutées et sur option les différents opérandes et les registres généraux modifiés.

* l'historique de l'exécution. L'utilisateur peut obtenir des renseignements sur les "N" dernières instructions exécutées (N ayant été défini par une requête préalable).

PILOTE rend possible ces opérations par les requêtes :

AT, STOP REFERENCE, STOP INSTRUCTION -	permettent de préciser les conditions qui provoqueront une suspension de l'exécution.
TRACE ON	active la prise en compte des informations nécessaires à la construction d'une trace des instructions simulées.
COLLECT ON "N"	construit de façon cyclique l'historique des "N" dernières instructions simulées.

Il est aussi possible de connaître le nombre d'instructions prise en compte dans le programme testé (requête COUNT), de choisir l'unité d'entrée-sortie qui recevra la trace et l'historique (requête PRINT) et de préciser quelle action on veut entreprendre lors de l'apparition d'une interruption asynchrone (requête ON). (référence 22, 26).

Une place particulière doit être donnée à la requête BREAK qui permet d'insérer des points d'arrêt qui ne nécessitent pas la simulation des instructions. En effet lorsque l'instruction se trouvant à l'adresse spécifiée doit être exécutée il se produit une interruption asynchrone du type erreur de programmation qui provoque l'arrêt de l'exécution et l'activation de l'environnement de mise au point.

Ainsi les outils dynamiques autorisent un contrôle continu du fonctionnement d'un programme ; les outils statiques quant à eux permettent un contrôle par point d'arrêt, les composants testés s'exécutant librement.

L'activation de la simulation et sa désactivation doit pouvoir se faire quand l'utilisateur le désire. Nous devons évoquer, en effet, les notions d'efficacité et d'économie : la simulation d'une instruction est environ cinquante à cent fois plus coûteuse que son exécution normale. C'est pourquoi il est indispensable que la simulation ne s'applique qu'à des portions très localisées du programme à mettre au point. Il est exclu que la totalité du programme soit simulée et, en conséquence, l'utilisateur peut définir (par des requêtes appropriées) les parties erronées qui doivent être soumises à la simulation.

Enfin nous devons citer que dans le cas où PILOTE est un environnement de mise au point disponible dans le système CMS, il fournit alors un troisième type d'outils que nous nommerons "outils de confort".

I-2-1-3 - "Outils de confort" de PILOTE

Ils ont pour but d'affranchir l'utilisateur de travaux fastidieux et de donner un meilleur aspect extérieur aux requêtes de PILOTE.

La souplesse d'utilisation d'un système de mise au point dépend en grande partie des options prises pour référencer la mémoire. Des emplacements mémoires doivent pouvoir être examinés et modifiés, et il doit être possible de poser des points d'arrêt dans une suite d'instructions. Chaque requête met en cause une adresse mémoire.

La manière la plus élémentaire et la plus naturelle de désigner une adresse est de la nommer par un nombre (hexadécimal dans notre cas). Toutefois, on peut rappeler que lorsqu'un programmeur réalise son algorithme au moyen d'un programme écrit en langage d'assemblage, par exemple, il est complètement affranchi du problème de l'adresse réelle en mémoire des quantités qu'il définit ou utilise. Ceci grâce à la facilité qui lui est apportée par l'utilisation de variables symboliques. Par contre au cours de la phase de mise au point, il est obligé de connaître très exactement l'emplacement de son programme dans la mémoire rapide du calculateur. Il semble donc logique de fournir la même facilité d'adressage symbolique et avec les mêmes symboles dans la phase de mise au point.

Nous devons convenir que l'utilisateur de PILOTE doit disposer au minimum de l'information qu'il a mise en place sur chaque variable lors de l'écriture de son programme (nature du contenu de cette mémoire, longueur de cette zone, etc...). La procédure adoptée se déroule en trois phases (référence 21).

* recueillir les tables de symboles de l'assembleur. Nous avons dit précédemment que PILOTE est plus spécialement conçu pour les composants écrits en langage d'assemblage. Au cours de son travail l'assembleur construit des tables de correspondance entre variables symboliques et les adresses qu'elles définissent. Cette première action nous donne des renseignements sur le contenu d'un emplacement mémoire, sa taille, et son adresse relative par rapport au début du programme. Ce dernier renseignement nous permettra, après chargement de ce composant, de calculer l'adresse réelle qui doit être associée au symbole correspondant.

* Mettre en forme ces tables. Il s'agit d'ordonner les quantités recueillies lors de la première phase afin de permettre une recherche rapide des informations associées à une variable symbolique.

* exploiter lors de chaque requête de PILOTE ces renseignements afin de calculer l'adresse réelle de l'emplacement mémoire référencé, afin de connaître le codage de l'information qu'il contient etc... (voir Fig. I.1).

Un problème particulier apparaît lorsqu'on utilise ce type d'adressage, appelé adressage symbolique, pour des programmes assemblés séparément puis chargés en mémoire et exécutés ensemble. En effet, certaines zones de mémoires bien que différentes peuvent être référencées par des noms identiques.

C'est pourquoi chaque symbole doit être qualifié par le nom du programme dans lequel il a été défini. De cette manière, d'une part les symboles d'un seul programme peuvent être disponibles à un instant donné et d'autre part le choix du programme utilisable peut être modifié à chaque instant. Ce principe de spécification du programme à sélectionner n'est pas une contrainte trop conséquente pour l'utilisateur.

Diverses opérations nécessaires a l'adressage symbolique

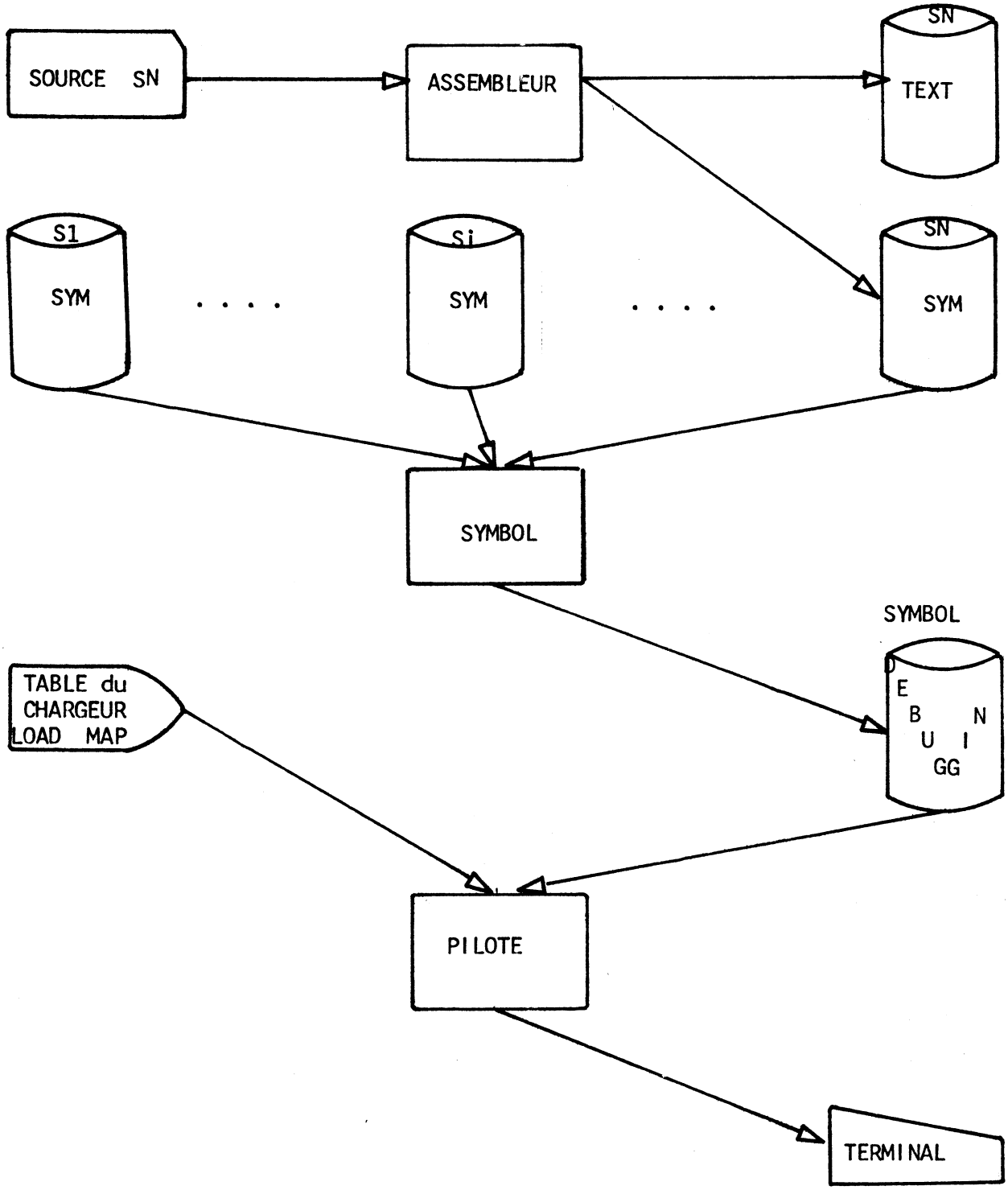


Fig. I-1

PILOTE donne ainsi la possibilité de référencer des emplacements mémoires de l'ensemble des programmes en cours de test de trois façons différentes :

* soit par une adresse absolue en hexadécimal → nombre hexadécimal suivi de .A

* soit par une adresse relative en hexadécimal → nombre hexadécimal suivi de .R

* soit par une adresse symbolique → chaîne alphanumérique suivie de .S

("par adresse relative" sous-entend un déplacement par rapport à une origine qui a été préalablement fixée).

Ces conventions sur les adresses s'accompagnent de facilités de codage de l'information. Le contenu d'une zone mémoire peut être donné soit sous la forme indiquée dans le programme source, soit sous une autre forme définie temporairement.

I-2-2 - QUELQUES PARTICULARITES DE PILOTE

Dans le paragraphe précédent, nous avons énuméré quelques uns des outils disponibles dans l'environnement conversationnel de mise au point PILOTE. Un certain nombre d'hypothèses ont directement influencé sa réalisation.

PILOTE appartient à la seconde classe des supports de mise au point disponibles dans CMS (la première est constituée d'environnements tels que FORTBUG etc...). C'est-à-dire qu'il est conçu pour des programmes écrits en langage d'assemblage lesquels représentent des fonctions d'un système en cours de test : par exemple, un programme superviseur d'entrée-sortie, un module de gestion des interruptions asynchrones (l'exécution de ce type de programme est rendue possible grâce aux particularités du système CMS qui admet que tout programme puisse travailler en mode maître).

Le programme testé et PILOTE utilisent des unités d'entrée-sortie. Il faut que chacun d'eux reçoive toutes les interruptions de fin d'opération d'entrée-sortie qui lui sont destinées et uniquement celles-là. Il se peut même que ces deux programmes utilisent les mêmes unités d'entrée-sortie ; par exemple le terminal peut être initialisé par un module en cours de test et être utilisé par

PILOTE pour lire des commandes. Pour qu'aucune des informations destinées au programme testé ne soient perdues, il est nécessaire que PILOTE, après avoir travaillé sur cette unité, la remette dans son état initial. Ceci nous amène à inclure dans PILOTE un superviseur d'entrée-sortie.

Le simulateur d'instructions doit tenir compte de tout ce qui peut modifier le déroulement séquentiel du programme qui lui est soumis. En particulier, les interruptions asynchrones perturbent ce déroulement, ce qui pourrait provoquer une désactivation du simulateur. Il faut donc que l'ensemble de ces phénomènes lui soit réfléchi afin qu'il puisse contrôler la suite de l'exécution. Il est aussi nécessaire que toute condition anormale provoque l'arrêt du programme en cours de test et la réactivation des outils statiques même si l'interpréteur n'était pas en fonction. Ceci nous amène à inclure dans PILOTE un module de gestion des interruptions.

Lors de la constitution d'un historique de l'exécution,PILOTE doit sauvegarder des informations intermédiaires. Les tables de symboles utilisées pour l'adressage symbolique ne peuvent résider en mémoire centrale. Ceci nous amène à inclure dans PILOTE un module de gestion de mémoire libre et un module de gestion de fichier.

Ces quelques exemples mettent en évidence le fait que très rapidement, on est conduit à associer aux fonctions de mise au point des fonctions utilitaires qui font d'un support tel que PILOTE un véritable système de programmation.

I-3 - PILOTE, SUPPORT GENERAL DE MISE AU POINT

Les fonctions inclues dans l'environnement PILOTE (superviseur d'entrée-sortie, gestion des interruptions, gestion de la mémoire libre) ont conduit à le modifier et à l'étoffer pour qu'il devienne, de par cette extension, un programme autonome capable de "conduire" un système de programmation quelconque, c'est-à-dire capable de supporter ce système.

La réalisation d'un tel composant standardise la mise au point car d'une part on dispose du même outil lors des deux phases distinctes de la mise au point -test des modules pris séparément et test de l'ensemble du système- et d'autre part on bénéficie d'un outil puissant et d'emploi commode pour effectuer la dernière étape de la réalisation.

Pour qu'une telle transformation soit effective il faut que PILOTE, ou tout autre support de mise au point de ce type, possède une propriété très importante :

Il doit être totalement indépendant du système testé, c'est-à-dire qu'il ne doit mettre en jeu aucune des fonctions de ce système, et que ce dernier doit totalement ignorer qu'il fonctionne sous la conduite d'un support de mise au point.

Cette propriété entraîne quelques remarques :

* Pour tester une fonction quelconque du système il ne faut pas que le support de mise au point utilise cette fonction. En effet, une erreur dans ce module partagé rendrait l'aide à la mise au point inopérant.

* Le cadre dans lequel le système devra évoluer, une fois terminé, ne doit pas être modifié car la suppression des programmes de mise au point qui auraient pu être inclus est susceptible de provoquer de nouvelles erreurs.

Cette transformation a été réalisée à plusieurs reprises pour permettre la mise au point de systèmes expérimentaux. Les modifications qui ont du être apportées à PILOTE environnement de CMS sont minimes. La procédure d'initialisation a été remplacée, les possibilités offertes par l'adressage symbolique ont été supprimées (en effet certaines fonctions de CMS, telle que la gestion de fichier, sont utilisées pour ranger les tables de symboles et pour les créer). Pour atteindre cet objectif nous avons été amené à résoudre un certain nombre de problèmes que nous exposerons dans le paragraphe I.4.2.

I-4 - ETUDE CRITIQUE DES OUTILS DE MISE AU POINT PRESENTES

Dans les deux paragraphes précédents, nous avons détaillé divers ensembles d'aide à la mise au point disponibles sur une machine virtuelle. Si nous souhaitons mettre au point un système implanté sur une telle machine, on peut se demander lequel de ces ensembles conviendra le mieux, quels sont les avantages et les inconvénients de chacun d'entre eux.

I-4-1 - AVANTAGES

Les fonctions consoles du système CP-67, l'environnement PILOTE dans CMS et PILOTE support général de mise au point, permettent tous trois une mise au point conversationnelle. Cette méthode de travail présente de nombreuses facilités pour l'utilisateur (référence 2, 22, 26). En particulier, elle fournit un gain de temps appréciable, elle donne à l'utilisateur la possibilité d'agir et de réagir en fonction de la situation présente et non plus en fonction d'une situation qu'il imagine à priori (chaque utilisateur à toujours tendance à affirmer que son programme fonctionne correctement), elle lui permet, enfin, de ne plus être inondé par une grande masse d'informations puisqu'il peut choisir sélectivement celles qui lui sont nécessaires.

Les avantages de la méthode conversationnelle ne sont pleinement ressentis que si l'utilisateur dispose d'outils élaborés qui lui fournissent une réponse rapide à chacune de ses questions, qui lui donnent la possibilité d'exercer un contrôle très fin sur ses programmes. Un interpréteur (ou simulateur) d'instructions satisfait les moindres désirs d'un utilisateur exigeant. Cette technique de mise au point doit toutefois être couplée à la possibilité de choisir à chaque instant, et en fonction des circonstances, les portions de programme qui doivent être simulées. PILOTE, environnement de CMS ou support général de mise au point, fournit un interpréteur et une série d'outils utilisant les renseignements fournis par celui-ci. Ce simulateur peut être désactivé à tout instant par l'utilisateur.

Nous avons vu également que pour être satisfaisant un ensemble d'aide à la mise au point doit être indépendant du système testé [illegible] fonctions consoles de CP-67, qui simulent le pupitre d'un ordinateur, satisfont à cette propriété. La conception et la réalisation de PILOTE a été fortement influencée par cette exigence.

PILOTE, composant de CMS, est bien adapté aux problèmes qui lui sont soumis. De part les facilités et les outils qu'il offre (interprétation, mise au point conversationnelle, adressage symbolique etc...), il permet un travail rapide, efficace et agréable.

I-4-2 - INCONVENIENTS

Les fonctions consoles du système CP-67, certes ne provoquent pas l'arrêt du calculateur réel, utilisent beaucoup de "temps système" (en particulier la fonction TRACE). Elles sont exécutées par le système CP-67 lui-même qui utilise ainsi du temps qui ne pourra être affecté à d'autres machines virtuelles. Bien que le problème de coût, soulevé par les homologues réels de ces opérations, soit moins critique, il existe toujours. On retrouve aussi un autre désavantage lié à ce type d'outils de mise au point :

l'ensemble des fonctions disponibles est difficilement extensible car il entraîne la modification du système CP-67.

Pour PILOTE, support de mise au point, de nombreux problèmes apparaissent lors de sa réalisation. Au paragraphe I-3, nous avons énoncé qu'un tel ensemble doit être indépendant du système testé et que ce dernier doit ingorer et la présence du support et le contrôle qu'il exerce. Les deux ensembles, système testé et support de mise au point, résident pourtant dans le même calculateur et doivent être présent en mémoire centrale de façon permanente et simultanément. Ceci implique au minimum que le système testé soit amputé d'une partie de la mémoire centrale dont il pourrait normalement disposer. Pour réaliser cette soustraction, deux solutions sont possibles :

* On peut tout d'abord remarquer que dans la plupart des cas, lors de la phase d'initialisation, un système calcule la taille de la mémoire dont il dispose. Cette opération consiste à rechercher la plus grande adresse valide. Pour implanter PILOTE en mémoire, nous devons faire croire au système que la mémoire centrale s'arrête là où est implanté le support de mise au point. En conséquence, il faut d'abord charger en mémoire ce dernier et ensuite, sous son contrôle , amener le système à tester. De même la phase d'initialisation devra être entièrement contrôlée.

Une réalisation du couple Système testé - PILOTE

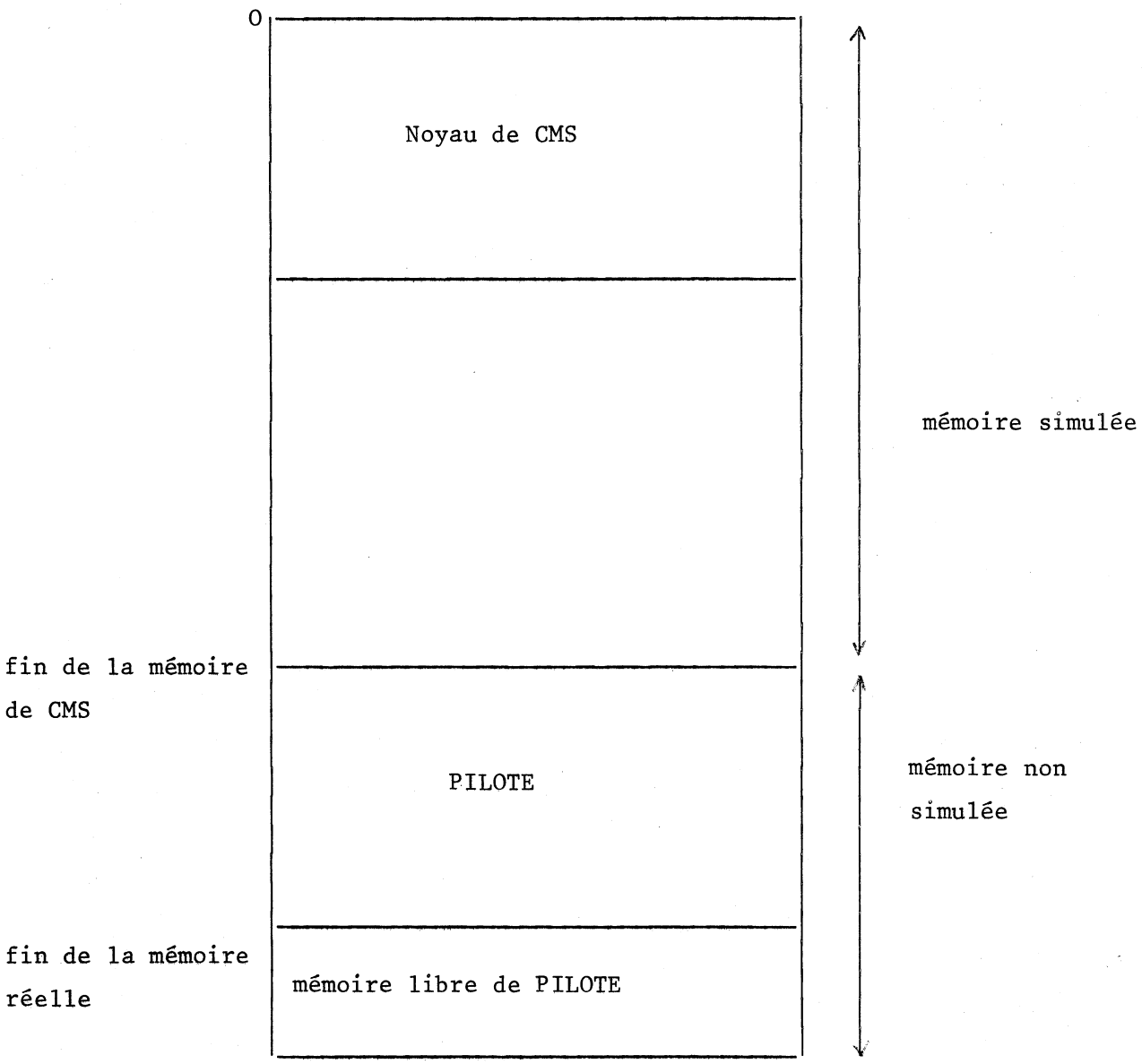


Fig. I.2.

Cette façon de procéder à le principal mérite de respecter l'indépendance entre les deux ensembles. Mais la séquence de chargement et d'initialisation du système à tester doit être entièrement simulée pour que ce dernier ne vienne pas "écraser" le support de mise au point.

Nous avons défini comme critère d'utilisation du simulateur, la présence d'erreur dans la partie du programme qui lui est soumis. Dans notre cas, cette interprétation fort longue des deux premières phases, ne se justifie plus par des impératifs de mise au point. En général, chargement et initialisation se déroulent correctement, la simulation n'est utilisée que pour donner des valeurs "correctes" à certains paramètres du système à tester.

* On peut aussi mettre en place d'abord le système testé et lui permettre de s'initialiser. Le chargement du support de mise au point s'effectue ensuite et celui-ci au cours de sa propre initialisation, modifie les paramètres de l'autre système pour lui indiquer que la taille de la mémoire principale a diminué. Cette opération doit être antérieure à l'utilisation de la mémoire et en particulier avant que le module de gestion de la mémoire libre n'est pris le contrôle.

Cette façon de faire se traduit par quelques modifications et du système testé et du support de mise au point : en d'autres termes, la phase d'initialisation de ce dernier devient spécifique du système testé. De plus, une connaissance approfondie de l'ensemble des deux programmes est indispensable. On constate alors qu'une certaine complicité entre les deux systèmes devient nécessaire et de ce fait les exigences de l'indépendance sont quelque peu transgressées.

Quelle que soit la méthode choisie on obtient une situation caractéristique représentée par la figure I-2.

L'initialisation des deux ensembles étant terminée, il faut activer le système testé ; pour cela deux options sont acceptées :

- ou bien la simulation est effective
- ou bien le système a un fonctionnement autonome.

Dans le premier cas, le simulateur doit avoir un contrôle absolu et en particulier il veut que toutes les interruptions asynchrones lui soient réfléchies. Ceci impose que la zone mémoire d'adresse 0 à 256 (zone privilégiée dans l'architecture de l'ordinateur 360 où l'unité centrale range certaines informations relatives aux interruptions en particulier (voir référence 17)) lui soit réservée et que par la même, il doit simuler cette zone pour le système testé (voir Fig. I-3).

Dans l'autre cas, le système testé fonctionne de façon autonome. La zone privilégiée que nous venons de décrire doit lui appartenir, il faut donc sauvegarder tout ce que le support de mise au point pouvait y avoir mis. Il convient aussi d'empêcher une destruction du support de mise au point par des instructions anarchiques.

Puisque d'une part PILOTE et le système en cours de test résident dans la même mémoire centrale ; puisque d'autre part aucun des deux ensembles exerce un contrôle quelconque sur l'autre, il faudrait confier à un procédé technologique le soin d'assurer la protection de PILOTE contre les actions intempestives du système testé. Pourtant, une solution telle que l'utilisation des clés de protection mémoire n'est pas envisageable. En effet, pour cela deux conditions devraient être remplies :

* le système testé ne doit pas utiliser la clé de protection affectée à PILOTE.

* PILOTE doit être réactivé chaque fois qu'une violation de cette clé est détectée.

Ces conditions ne peuvent être satisfaites puisque, à priori, nous ne connaissons pas le comportement du système et en particulier les clés de protection mémoire qu'il utilise et puisque la zone privilégiée (zone d'adresse 0 à 256) qui permet, sur interruption ou condition anormale, d'activer le composant de traitement correspondant, appartient au système testé.

Ce problème de protection, qui est fondamental, ne peut être résolu dans le cadre actuel que par des solutions de compromis.

Problème de la zone privilégiée

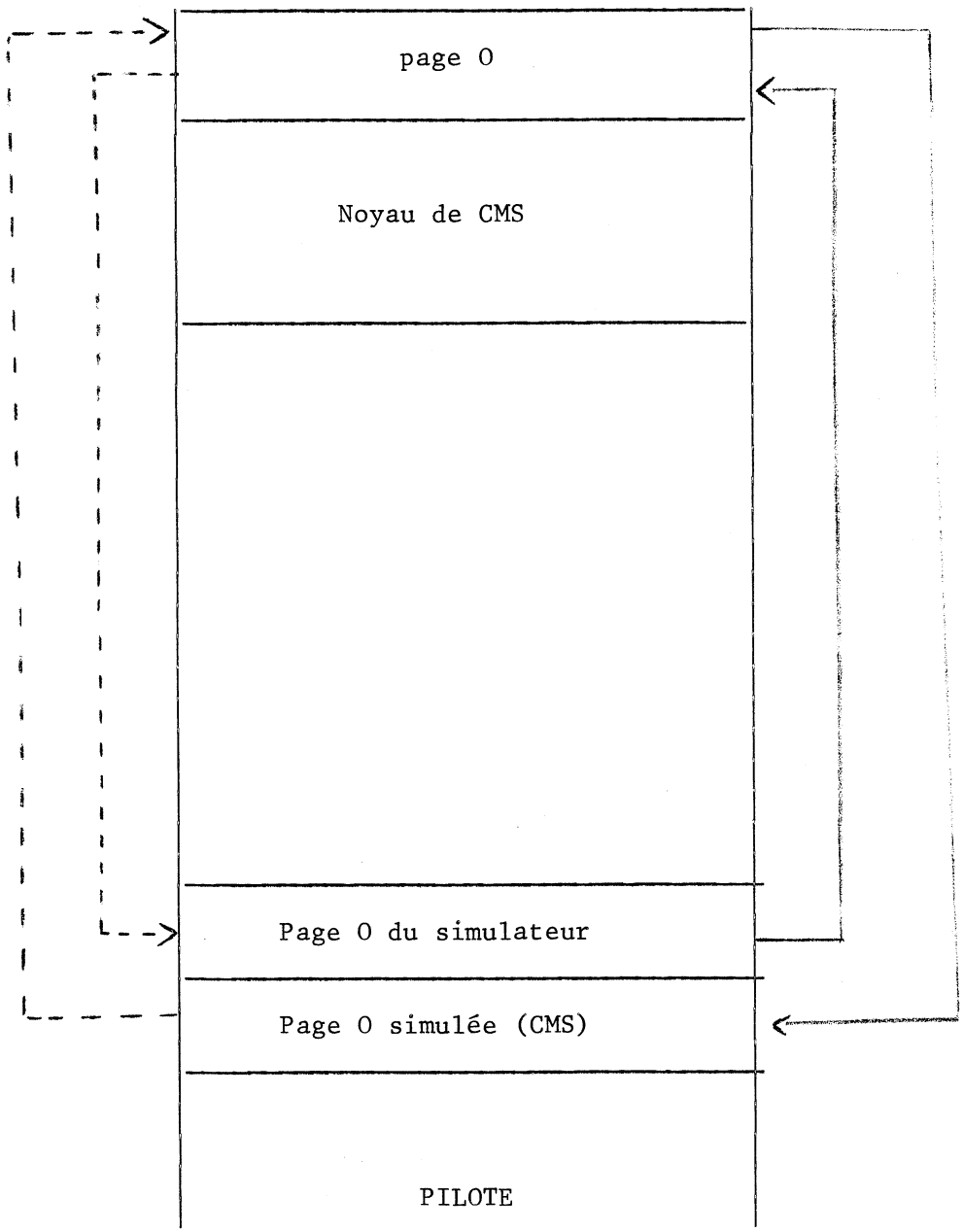


Fig. I-3

I-5 - CONCLUSION

Le système CP-67 offre un premier ensemble d'outils de mise au point permettant de travailler sur un système de programmation quelconque, mais certains défauts relatifs à la "mise au point pupitre" sont toujours présents. On peut citer en particulier la non possibilité d'extension du jeu d'outils existants.

PILOTE, composant du système CMS, fournit une aide efficace lors de la première phase de la mise au point : les tests sur les modules du système pris séparément à l'aide des outils statiques, dynamiques et à l'adressage symbolique qui sont présents dans cet ensemble.

PILOTE, support de mise au point, soulève quant à lui, des problèmes non négligeables qui diminuent de beaucoup sa souplesse d'utilisation.

Mais les deux versions de PILOTE ont un problème commun à résoudre : la protection de l'aide à la mise au point contre les actions erronées du système testé.

CHAPITRE - II

DESCRIPTION D'UN SYSTEME DE CONTROLE : SPY

Le système CP-67 génère des machines virtuelles, images fidèles de calculateurs réels. Le programmeur dispose d'un cadre privilégié où il peut créer, réaliser et mettre au point de nouveaux systèmes de programmation. Un certain nombre d'ensembles d'aides à la mise au point lui apportent des facilités pour cette élaboration.

Mais ces outils très efficaces, ne sont pas implantés dans un environnement disponible à chaque instant et qui n'assure pas leur protection contre les erreurs de programmation. Ils ne s'adaptent pas à un système de programmation quelconque.

Nous avons conçu et créé un système général de mise au point et nous y avons intégré les outils déjà existants. Pour cela nous avons défini la notion de système de contrôle, la notion de machine Espion et réalisé le système SPY.

II-1 - DEFINITION D'UN SYSTEME DE CONTROLE

Nous appelons système général de contrôle d'un ordinateur en activité (ou simplement système de contrôle) un ensemble de fonctions permettant de tester un système de programmation quelconque. Tester signifie que nous entendons diriger ou participer à toutes les actions qui ont pour but de produire un système fiable et qui donne de bonnes performances.

Un système de contrôle doit d'une part fournir des outils que l'on puisse adapter instantanément à tout ensemble de programmation existant ou futur. Ces aides à la mise au point doivent comprendre à la fois un groupe de fonctions de base et un mécanisme d'extensions. D'autre part, il doit définir un environnement où les outils de mise au point seront protégés de toutes les erreurs du programme en cours de test et où ceux-ci pourront travailler avec un maximum d'efficacité.

Décrivons les méthodes de mise au point qu'un utilisateur peut souhaiter trouver dans un système de contrôle, et les problèmes soulevés par la réalisation d'un tel ensemble.

II-1-1 - Environnement conversationnel

Un système de contrôle doit être un système conversationnel.

Cette méthode de travail présente de nombreux avantages pour l'utilisateur (référence 26). En particulier, elle fournit un gain de temps appréciable, elle donne à l'utilisateur la possibilité d'agir et de réagir en fonction de la situation présente.

Dans notre cas particulier ces facilités sont encore renforcées par la complexité et la grande taille des programmes à tester. Qui peut envisager d'avoir une trace complète de toutes les instructions exécutées par le système de programmation OS/360 au cours d'une journée ? Il ne s'agit plus, maintenant, de trouver une erreur de programmation qui apparait lors de l'exécution d'une instruction, mais de trouver un défaut dans un algorithme, lorsque le système se trouve dans un état particulier.

En effet on ne peut demander à un utilisateur de définir à priori toutes les situations qui peuvent se présenter à lui, de mettre en place des jalons qui lui permettront de découvrir celles non prévues, et de mettre en place des outils pour vérifier le bon fonctionnement de son système.

En général, sur une erreur, les conditions exactes dans lequel se trouve le système testé sont difficiles à reproduire, il faut donc que l'utilisateur puisse intervenir immédiatement en fonction d'une situation précise.

Un environnement conversationnel lui donne la possibilité de faire une étude approfondie de son programme et de ne plus se limiter à un dépouillement d'informations. Le système de contrôle lui fournit une aide réelle, au lieu de le noyer sous des monceaux de résultats.

II-1-2 - Langage de commandes

Il va de soi que tout environnement conversationnel doit posséder un langage de commande suffisamment évolué pour dégager l'utilisateur de toutes contraintes. Il devient alors nécessaire de construire un langage souple d'emploi mais qui possède un lot de commandes simples, un ensemble de mots clés significatifs, et surtout un mécanisme d'extension qui permette d'adapter la mise au point pour faire face à toutes situations particulières.

Grâce à la puissance des outils, l'utilisateur peut se consacrer uniquement à l'étude détaillée de ses algorithmes.

II-1-3 - Les mécanismes de mise au point

Le langage de commandes donne accès à des outils de mise au point. Rappelons qu'il est souhaitable de disposer de deux classes d'aides à la mise au point : des outils statiques et des outils dynamiques (référence paragraphe I-2-1).

L'un des mécanismes de base du système de contrôle est un simulateur (ou interpréteur) d'instructions. Dans notre cas il est plus juste de dire que l'on dispose d'une unité centrale simulée. En effet, le simulateur interprète les

instructions des programmes qui lui sont soumis, mais aussi il réfléchit à ce dernier les interruptions asynchrones qui lui sont nécessaires, il simule les registres généraux utilisés par le programme etc...

Nous pouvons dire alors que cette unité centrale simulée, qui fournit un résultat absolument identique à celui produit par une exécution réelle, donne la possibilité d'un contrôle minutieux du système testé, mais provoque une forte distorsion de l'échelle des temps (le temps d'exécution d'une opération simulée est plusieurs dizaines de fois supérieur à celui de la même opération exécutée normalement). En complément, il est nécessaire de donner au système testé la possibilité de s'exécuter de façon autonome.

II-1-4 - Indépendance vis à vis du système testé

Si l'on veut que l'ensemble des aides à la mise au point décrit précédemment soit utilisable en toute circonstance, il faut les implanter dans un cadre privilégié.

Essentiellement aucune supposition sur le fonctionnement du système testé ne doit être faite et tout algorithme doit pouvoir être testé. De plus les aides à la mise au point ne doivent modifier en rien le système testé. Ce cadre doit rendre le système de contrôle parfaitement "transparent", c'est-à-dire que le système en cours de test ignore totalement les contrôles dont il faut l'objet.

En résumé une parfaite fiabilité du système de contrôle est indispensable, et il doit posséder une protection totale contre les erreurs du système testé ; celui-ci doit pouvoir s'exécuter dans les conditions réelles de son fonctionnement futur.

II-2 - CONSTRUCTION D'UN SYSTEME DE CONTROLE

Les propriétés classiques que nous avons imposées au langage de commande et aux mécanismes de base, la présence d'un environnement conversationnel, se retrouvent dans tout ensemble d'aide à la mise au point (aux références 2 et 26 se trouve une liste non exhaustive d'ensembles de mise au point).

La dernière classe de propriétés : indépendance et transparence, est plus difficile à mettre en oeuvre. Elle pose deux problèmes, d'une part celui de la protection du système de contrôle et d'autre part, celui de la non modification du cadre où évolue le système testé.

II-2-1 - Protection

Le système testé peut initialiser toutes les opérations possibles sur la machine où il réside, y compris les opérations dites "privilégiées" (c'est-à-dire qui modifient l'état du calculateur). Les causes d'erreurs sont innombrables et imprévisibles, donc le système de contrôle ne doit accorder aucune confiance au système testé. La protection contre le mauvais fonctionnement de ce dernier doit être totale et rigoureuse.

Néanmoins le système de contrôle doit pouvoir agir sur le système testé afin de découvrir les raisons d'un mauvais fonctionnement.

A priori ces deux contraintes semblent être en contradiction, puisque nous affirmons que d'une part le système de contrôle doit être parfaitement isolé et que d'autre part il doit avoir des contacts avec le milieu extérieur (essentiellement le système en cours de test).

Considérons deux programmes en cours d'exécution (nous les appelerons tâches).

Les règles de protection entre deux tâches (référence 5) nous permettent d'affirmer :

- qu'aucune des deux tâches ne doit pouvoir lire, modifier ou détruire des informations appartenant à l'autre tâche si celle-ci ne l'autorise pas.

- si l'une des deux tâches a un fonctionnement défectueux, elle ne doit pas affecter l'exécution de l'autre tâche.

Plus généralement, si nous appelons ressource toute entité utilisée par une tâche pour continuer son exécution, nous dirons qu'aucune des deux tâches ne peut utiliser une ressource de l'autre tâche, quelque soit les conditons d'exécution, si elle n'y a pas été autorisée.

Considérons maintenant une tâche D, représentant le système de contrôle, et une tâche X représentant le système testé. Les relations entre ces deux tâches sont partiruclières. Il faut naturellement que toute ressource de D soit protégée des actions de X. C'est-à-dire que X ne doit pas être autorisée à utiliser les ressources de D. Mais il faut que D puisse utiliser toutes les ressources de X. Si cette condition n'est pas remplie alors D est inutilisable pour localiser une erreur dans X.

Nous voulons aussi que D soit capable de fixer un ensemble de conditions initiales quelconques et de telle façon que l'exécution de la tâche X puisse commencer en un point quelconque. Cette relation est nécessaire si l'on veut que D soit capable de fixer un état de X qui a provoqué un mauvais fonctionnement.

En résumé, nous voulons que la tâche D ait la possibilité :

* dresser la liste des ressources de la tâche X
* examiner chacune des ressources de cette tâche
* modifier l'état ou fixer un nouvel état pour chacune de ces ressources.
* initialiser l'exécution de X en un point quelconque avec l'état correspondant
* utiliser des ressources qui lui soient propres.

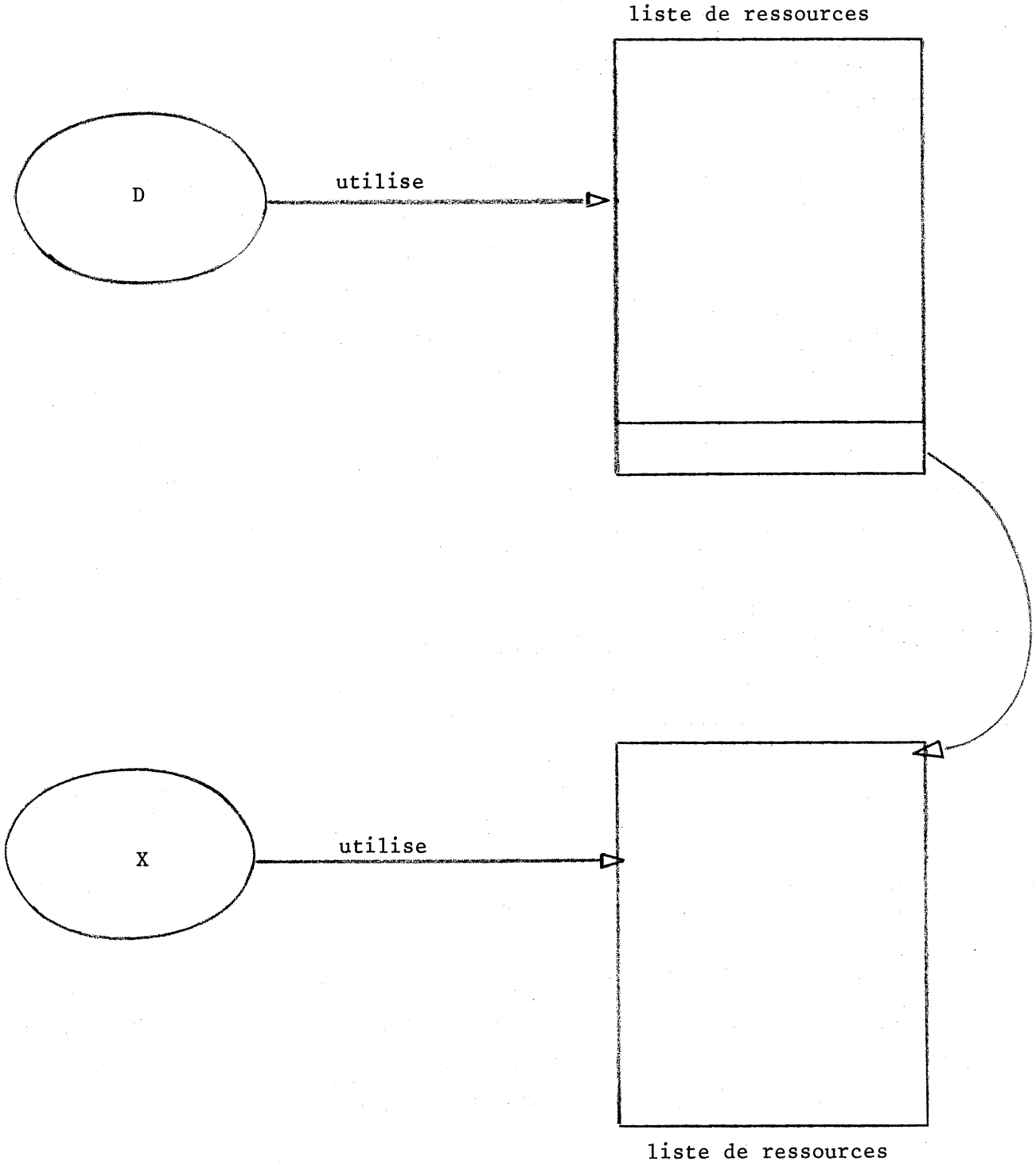


Fig. II.1 - Description des ressources de D et X.

Si l'on suppose qu'à chaque tâche est associée une liste des ressources qu'elle est en droit d'utiliser, les ressources de la tâche X doivent figurer et dans la liste de la tâche X et dans celle de la tâche D. Cette dernière peut aussi posséder des ressources qui ne soient pas des ressources de X. Nous obtenons ainsi une hiérarchie représentée par la figure II.1.

La tâche D peut utiliser les ressources de X qui se trouvent naturellement dans sa liste de ressources, quant à X elle ne peut qu'utiliser ses ressources.

La tâche D (système de contrôle) doit avoir la possibilité de créer une nouvelle liste de ressources et d'affecter cette liste à la tâche X. Ceci est très important car il met en évidence le fait que pour résoudre nos problème de protection il faut donner une propriété particulière à D, mais que ce sont les règles classiques de protection qui seront mises en jeu après. Il est donc inutile d'incorporer à D un nouveau mécanisme de protection, simplement D possède une ressource particulière qui est une nouvelle liste de ressources.

II-2-2 - Contrôle de l'exécution

Considérons la tâche X. Son exécution consiste en l'initialisation d'opérations élémentaires. Ainsi si X représente un programme Fortran en cours d'exécution, X initialise des opérations élémentaires qui sont des instructions Fortran. Pour ce faire X s'adresse à un "niveau Fortran" qui établit la correspondance entre les instructions Fortran et les opérations élémentaires reconnues par la machine.

Supposons que X entre dans une phase critique, elle a besoin d'utiliser un ensemble d'opérations plus étendu que celui dont elle disposait. En effet, cet ensemble doit comporter d'une part les opérations élémentaires utilisées jusqu'à présent et d'autre part des opérations élémentaires de mise au point.

Donc il faut intercaler entre la machine et la tâche X un niveau supplémentaire qui soit capable de reconnaître les opérations de mise au point. Nous obtenons une situation hiérarchisée représentée par la figure II.2.

Le niveau de mise au point est capable de reconnaître toutes les opérations élémentaires reconnues par le niveau 1 plus une série d'opérations élémentaires spécifiques à la mise au point.

Cette situation permet d'autre part au niveau de mise au point de contrôler toutes les conditions exceptionnelles destinées à X. Ainsi ce niveau peut être présent lorsque X a un fonctionnement erroné (correspond à l'initialisation d'une opération élémentaire inconnue par exemple), mais d'autre part toute condition initialisée par la machine (interruption asynchrone par exemple) est d'abord réfléchie au niveau mise au point qui décidera s'il doit la transmettre à X.

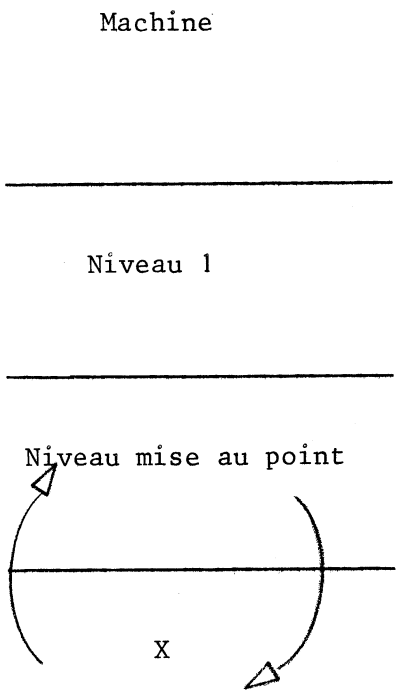


Fig. II.2

Le niveau de mise au point peut aussi disposer de facilités lui permettant d'arrêter l'exécution de X lorsqu'il lui semble qu'une séquence anormale d'opérations élémentaires est initialisée.

II-2-3 - Indépendance

Les solutions apportées au problème de la protection et à celui du contrôle de l'exécution permet d'affirmer que

- le système de contrôle n'aura aucune interférence sur le système testé. C'est-à-dire qu'aucune modification ne doit être apportée à ce dernier pour l'adapter à l'environnement de mise au point, qu'aucune modification à l'état du système testé est provoquée par l'exécution du fonction de mise au point.

- une erreur apparaissant dans le système de contrôle lui-même ne peut affecter le système en cours de test.

En conséquence le système "cobaye" ignore complètement les contrôles dont il fait l'objet.

II-3 - DESCRIPTION DU SYSTEME SPY

Le système de contrôle SPY a été développé sous le système de programmation CP-67, c'est-à-dire qu'il est disponible pour un utilisateur d'une machine virtuelle.

Sa réalisation a été faite en deux étapes :

* Définition d'un ensemble d'outils de mise au point et des mécanismes de base nécessaires à ceux-ci. Le travail a été réalisé par B. PETEUL (Ref. 26).

* Définition de propriétés qui permettent la protection totale du système de contrôle et le contrôle de l'exécution du système testé. Dès lors, il a fallu adapter les aides à la mise au point pour qu'ils puissent fonctionner avec ces nouvelles conditions.

La description des outils de test est décrit dans l'ouvrage cité à la référence 26, les modifications qu'il a fallu leur apporter pour les insérer dans le système SPY sont décrites aux chapitres III et IV.

Notre but a été essentiellement de définir les propriétés données précédemment (protection et contrôle) et de les mettre en oeuvre dans le cadre particulier du système CP-67.

II-3-1 - Définition d'une machine ESPION

Le système CP-67, générateur de machines virtuelles, fournit à chaque utilisateur des calculateurs virtuels totalement indépendants les uns des autres (référence 14, 15). Chaque utilisateur dispose de la configuration machine qui lui convient le mieux, et il peut y activer le système de programmation de son choix. Dès lors ce système ne peut pas utiliser des quantités virtuelles attachées à d'autres machines virtuelles. C'est-à-dire que ce systène ne peut faire référence qu'à la mémoire virtuelle du calculateur où il réside, ne peut qu'utiliser les unités d'entrée-sortie qui sont associées à cette machine etc... Ces affirmations restent valables même si le système a un fonctionnement défectueux.

Les mécanismes de protection du système CP-67 assurent une parfaite indépendance entre deux machines virtuelles. Nous avons voulu définir une nouvelle relation entre deux machines virtuelles : relation que nous appelerons de façon générale couplage de deux machines virtuelles.

Considérons deux machines virtuelles quelconques. Lors de leur création un certain nombre de ressources leur sont affectées (une unité centrale, une mémoire virtuelle, des unités d'entrées-sorties...), éventuellement certaines ressources complémentaires peuvent leur être attribuées lorsqu'elles le demandent. Nous donnons à l'une des deux machines la propriété fondamentale de pouvoir augmenter dynamiquement, et en cours de fonctionnement, le nombre de ses ressources au détriment de la seconde machine.

Ces ressources ne sont pas perdues pour la seconde machine, simplement nous donnons à une machine virtuelle, avec le consentement de l'autre, la possibilité de connaître et d'utiliser des ressources qui ne lui appartenaient pas au moment de sa création. Cette addition peut porter sur la mémoire centrale, sur l'unité centrale et sur les unités d'entrée-sortie.

D'après les propriétés du système CP-67 on peut affirmer que la machine supérieure (elle a augmenté le nombre de ses ressources) peut utiliser et ses ressources et celles de l'autre machine, qui, quant à elle ne peut utiliser que ses propres ressources. Ainsi, par exemple, un programme résidant dans la mémoire de la machine supérieure peut utiliser toute la mémoire (y compris celle qui a été ajoutée), mais un programme implanté dans la seconde machine ne peut référencer la mémoire appartenant à la machine supérieure.

La définition de cette relation nous place dans une situation équivalente à celle décrite par la figure II.1.

En fait, il est évident que la relation que nous avons appelée couplage est une relation d'inclusion entre deux machines virtuelles. On dira que la machine supérieure inclut la seconde machine (la seconde machine est incluse dans la machine supérieure).

II-3-1-1 - Applications de la relation d'inclusion

Cette relation définie entre deux machines virtuelles permet à l'une d'entre elles d'avoir accès à ce qui appartient à l'autre sans pour autant être obligée de mettre en commun ce qui lui est propre. Si l'on considère deux machines quelconques, elles sont indépendantes l'une de l'autre ; la relation d'inclusion supprime l'indépendance dans un sens. Une machine est indépendante de l'autre, laquelle n'est plus indépendante de la première.

Cette propriété semble résoudre quelques uns des problèmes soulevés par la réalisation d'un système de contrôle. En effet, prenons une première machine virtuelle où réside un système de contrôle et une seconde où est implanté un système de programmation. Nous couplons ces deux machines de façon à ce que celle où réside le système de contrôle inclut l'autre machine.

Nous disposons alors d'un ordinateur entier pour développer des outils de mise au point et les y faire fonctionner. L'indépendance entre les deux systèmes est réalisée à un niveau inférieur puisqu'elle devient une propriété de liaison entre deux calculateurs.

La relation de couplage peut porter sur la mémoire, sur les unités d'entrée-sortie mais aussi sur les unités centrales. L'unité centrale de la machine supérieure inclut l'unité centrale de la seconde machine. Toute opération élémentaire qui doit être exécutée par l'unité centrale incluse, est en fait soumise à l'unité centrale de la machine supérieure. Celle-ci peut reconnaître un plus grand ensemble d'opérations élémentaires. Les conditions exceptionnelles qui devaient être réfléchies à l'unité centrale incluse sont envoyées à la machine supérieure.

Nous avons ainsi créer une situation (voir fig. II.2) qui permet le contrôle de l'exécution par la machine supérieure.

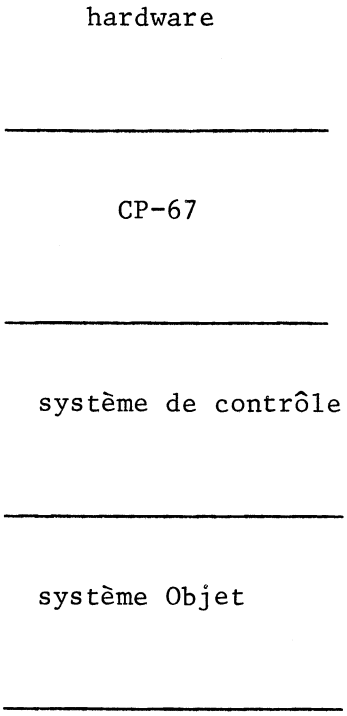


Fig. II.2

II-3-1-2 - Définitions

On appelle machine ESPION une machine virtuelle où est implanté un système de contrôle capable de diriger et de surveiller le fonctionnement d'une autre machine virtuelle.

Cette dernière est dite machine OBJET.

Lorsque ces deux machines seront couplées nous dirons que la machine OBJET est incluse dans la machine ESPION.

La relation d'inclusion fournit l'indépendance dont a besoin un système de contrôle et permet à celui-ci de contrôler l'exécution du système testé (il est implanté dans la machine OBJET).

II-3-2 - Définition du couplage

Le système SPY activable sur une machine ESPION peut contrôler l'exécution d'un système de programmation se trouvant sur une autre machine virtuelle.

Le système CP-67 possède de nouvelles fonctions qui donnent la possibilité de réaliser la relation d'inclusion et par la même il génère des machines ESPION. De ce fait, les outils de mise au point peuvent être considérés comme un système de programmation implanté sur une machine ESPION. L'ensemble création d'une machine ESPION, et aides à la mise au point forment le système de contrôle SPY.

Un programmeur, responsable de tests sur un système de programmation, travaillant sur une machine virtuelle, peut demander à n'importe quel moment, et quelque soit l'état de son calculateur virtuel, l'assistance du système de contrôle SPY. Pour cela le programmeur émet une demande explicite et le couplage automatique entre sa machine virtuelle, qui devient une machine OBJET, et une machine ESPION, où résident des outils de mise au point, est réalisé.

La relation qui lie une machine ESPION à une machine OBJET est une relation d'inclusion. C'est-à-dire que toutes les ressources affectées à la machine OBJET appartiennent aussi à la machine ESPION. Cette dernière possède aussi d'autres ressources qui lui sont propres -elles ne sont pas connues de la machine OBJET-. Le couplage porte sur les trois types de ressources dont dispose un ordinateur : unité centrale, unités d'entrée-sortie, mémoire rapide, et prend pour chacune d'elles une forme particulière.

Dans tous les cas la définition du couplage a été influencée par la nécessité qu'il y a de pouvoir le modifier à chaque instant. Ces modifications peuvent être très diverses :

* Demande de couplage sur une classe de ressources particulières ou abandon du couplage.

* Abandon définitif de l'aide fournie par le système SPY (sans destruction de la machine OBJET).

* Couplage conditionnel sur une classe de ressources.

Nous aurons toujours une bascule activation du couplage-désactivation momentanée du couplage. Cette nécessité de pouvoir activer ou désactiver le couplage entre deux machines virtuelles est souhaitable pour les deux raisons suivantes :

* si le programmeur souhaite utiliser les opérations élémentaires de mise au point il active le couplage. Dès lors il dispose des outils statiques et des outils dynamiques (voir paragraphe I-2-1). Il peut mettre en fonction un simulateur d'instructions, des opérations d'examen de sa machine virtuelle etc... Toutes les opérations initialisées par le programme implanté dans son calculateur virtuel seront exécutées par la machine ESPION.

* par contre, si temporairement, l'utilisateur ne désire pas utiliser les opérations élémentaires de mise au point il est inutile de faire appel à la machine ESPION. Il est souhaitable à ce moment là de désactiver temporairement le couplage, tout en imposant que toute condition anormale soit réfléchie à la machine ESPION. Ceci aura pour conséquence d'activer à nouveau le couplage. Cette possibilité est réalisable puisque la machine ESPION est placée "entre la machine réelle et le système testé".

II-3-2-1 - Inclusion de mémoires

Toute instruction de la machine ESPION peut référencer toute quantité de la machine OBJET alors que le contraire est impossible.

Décrivons la notion de mémoire virtuelle et définissons deux mémoires virtuelles incluses (référence 4,5).

Soit R et L deux ensembles ordonnés de blocs de longueur fixe (appelés généralement pages) qui constituent d'une part la mémoire du calculateur réel disponible -R-, et d'autre part la mémoire secondaire -L- emplacement utilisable sur les différents niveaux de la hiérarchie mémoire (tambours et disques par exemple).

Par définition on a la relation suivante

$R \cap L = \emptyset$ (ensemble vide)

ce qui est équivalent à dire que L est une extension de la mémoire réelle R.
Soit MV un ensemble de pages représentant une mémoire virtuelle.
On définit alors une transformation f qui applique MV dans R ∪ L. D'après la définition de R et de L si k est une page de MV alors f(k) appartient soit à l'ensemble R soit à l'ensemble L. On peut donc écrire

$f : MV \rightarrow MR \in R \cup L$

On appelle MR le lieu instantané de résidence de la mémoire virtuelle MV.
On définit maintenant une fonction Γ , appelée translation dynamique d'adresse, telle que :

si k est une page appartenant à MV alors
$\Gamma(k) = f(k)$ si f(k) est un élément de R
indéfinie si f(k) est un élément de L.

Cette fonction s'applique à une mémoire virtuelle. En effet, si on nomme x "adresse virtuelle" alors la fonction Γ lui fait correspondre une "adresse réelle" y si la page de la mémoire virtuelle contenant l'adresse x est en mémoire réelle (Rappelons que l'on peut décomposer une adresse générée par un programme en un numéro de page k et un déplacement dans cette page).

Si pour cette adresse x la fonction Γ prend une valeur indéfinie, la page de la mémoire virtuelle contenant x est en mémoire secondaire. Dès lors il faut redéfinir le lieu instantané de résidence de la mémoire virtuelle MV de telle sorte que la fonction Γ prenne une valeur définie.

Pour que deux mémoires virtuelles MV1 et MV2 soient disjointes il faut et il suffit que les lieux instantanés de résidence de ces deux ensembles soient disjoints. C'est-à-dire

$MV1 \cap MV2 = \emptyset \iff MR1 \cap MR2 = \emptyset$

Donnons maintenant la définition de la relation d'inclusion pour deux mémoires virtuelles.

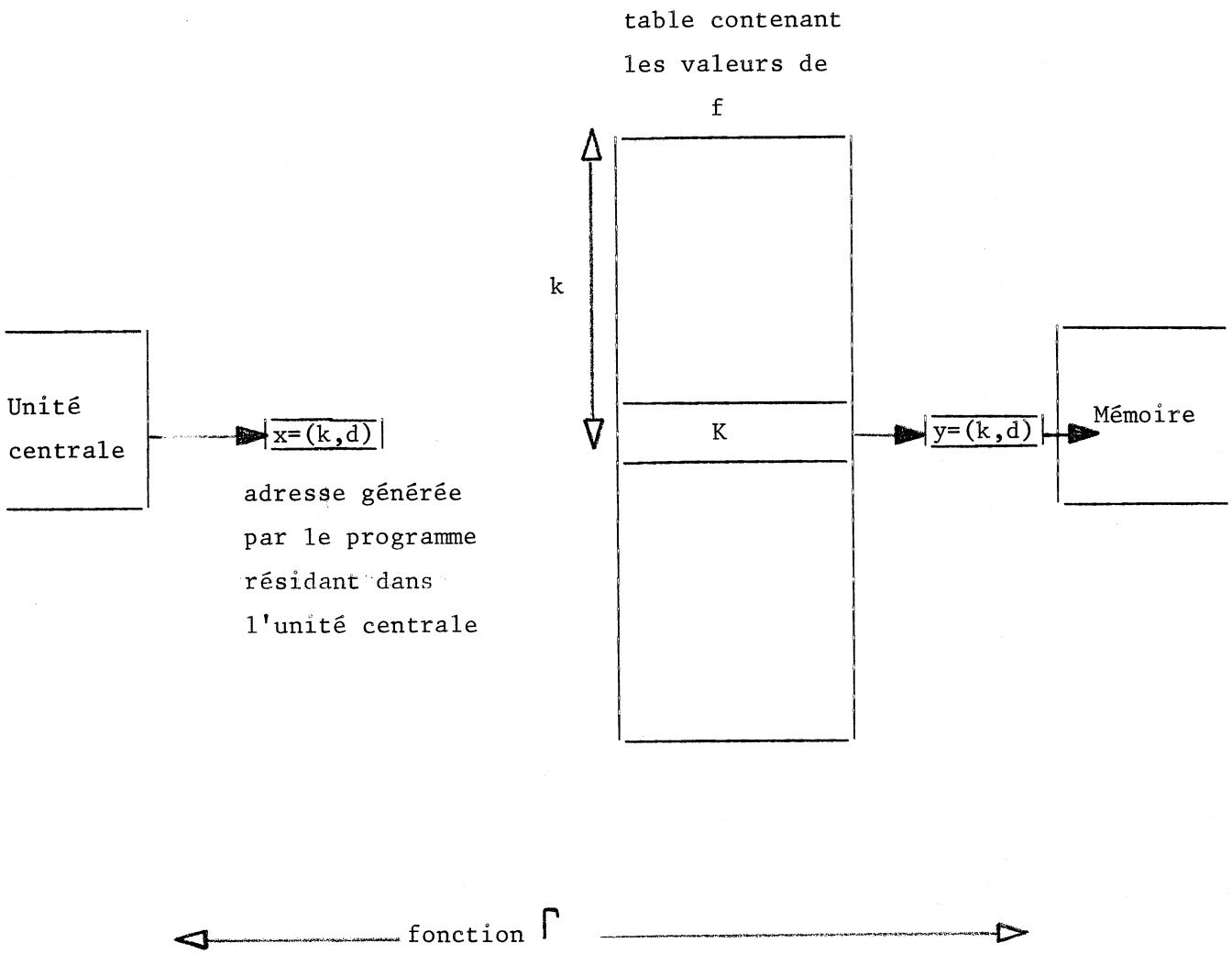


Fig. II.3 Translation d'adresse.

MV1 $\supset$ MV2 si à partir d'un certain élément k de MV1 le lieu instantané de résidence de MV1 coïncide avec le lieu de résidence de MV2.

Il est donc évident que l'on peut diviser MV1 en deux sous-ensembles disjoints MVS et MVO tel que

MVS $\cup$ MVO = MV1

MVO = MV2.

Il faut maintenant vérifier que cette définition répond aux objectifs que nous nous sommes fixés.

La définition d'une mémoire virtuelle permet de construire un mécanisme de protection de cette mémoire. En effet un programme résidant dans une mémoire virtuelle ne peut référencer que les informations se trouvant dans celle-ci. La transformation f fait correspondre à une adresse virtuelle générée par ce programme l'adresse réelle correspondante. Cette adresse réelle permet de référencer une information se trouvant dans le lieu instantané de résidence de cette mémoire. Il est donc impossible à ce programme d'utiliser des informations inaccessibles au moyen de la transformation f. La fonction Γ qui est appliquée à chaque adresse virtuelle contrôle cette protection.

Dès lors il est évident qu'un programme implanté dans MV1 peut référencer des quantités dans MVS et dans MVO donc des quantités dans MV2, alors qu'un programme résidant dans MV2, ne peut référencer que des éléments de MV2.

L'activation de cette relation de couplage entre deux mémoires virtuelles peut être réalisée très simplement.

La transformation f_1 associée à MV1 permet de déterminer le lieu instantané de résidence de cette mémoire. Pour la mémoire MV2 il existe de la même façon f_2. Coupler deux mémoires virtuelles, est réalisé en faisant coïndicer MVO et MV2. Pour celà il suffit de redéfinir les valeurs de la fonction f_1 de telle sorte que : (ceci donne une nouvelle transformation f_c)

si x est une adresse virtuelle alors

$f_c(x) = f_1(x)$ si x est une adresse virtuelle dans MVS

$= f_2(x)$ si x est une adresse virtuelle dans MVO, donc dans MV2.

La désactivation du couplage quant à elle peut aussi être réalisée par une redéfinition de f_c (nouvelle transformation $f_{c'}$).

$f_{c'}(x) = f_1(x)$ si x est une adresse virtuelle dans MVS

= indéterminée si x est une adresse virtuelle dans MVO.

De même on redéfinit f_2 de sorte que :

$f_2(x) = f_c(x)$ si x est une adresse virtuelle dans MVO.

Nous citons deux propriétés qui découlent de la définition de la relation d'inclusion :

- la mémoire MV2 peut être initialisée avant de réaliser le couplage, puisque les valeurs de la fonction f_2 sont utilisées pour construire la nouvelle transformation f_c. Il en est de même pour le sous-ensemble MVS de MV1. Par contre ce qui se trouvait dans le second sous-ensemble est détruit puisque les valeurs de f_1 s'y rapportant sont perdues. Les modifications faites dans MV2 lorsque le couplage est actif ne seront pas perdues puisque f_2 est redéfinie à partir des valeurs de f_c.

- la relation d'inclusion est itérative. Rien n'empêche la mémoire MV1 d'être à son tour incluse dans une mémoire virtuelle MV11. Le processus de construction du couplage est exactement le même. Il faut néanmoins tenir compte, dans ces itérations, de la taille maximum permise pour une mémoire virtuelle.

II-3-2-2 - Inclusion d'unités centrales

L'inclusion des mémoires virtuelles telle qu'elle vient d'être décrite implique que la machine virtuelle, destinée à être machine OBJET, ait été créée avant que le couplage soit demandé. Nous savons aussi que la mémoire virtuelle du calculateur OBJET peut avoir été initialisée, donc l'unité

centrale peut avoir déjà exécutée des instructions résidant dans cette mémoire. Tout ceci met en évidence le fait, qu'au moment du couplage, l'unité centrale de la machine OBJET se trouve dans un certain état qu'il faut respecter.

Affirmer que l'unité centrale de la machine OBJET est incluse dans celle de la machine ESPION, est équivalent à dire que l'unité centrale de la machine ESPION prend le contrôle de l'autre. La fonction essentielle de l'inclusion est de permettre à l'utilisateur de disposer d'un jeu plus étendu d'opérations élémentaires (opérations disponibles sur la machine OBJET plus les opérations de mise au point). Ces nouvelles possibilités autorisent le contrôle de l'exécution du système résidant dans la machine OBJET.

Quand le couplage est effectif, l'unité centrale de la machine ESPION a deux rôles à jouer :

* permettre aux fonctions de mise au point de s'exécuter
* permettre au système testé qui se trouve dans la machine OBJET de s'exécuter.

La réalisation de cette forme de couplage est guidée par une contrainte impérative : il faut préserver l'état de la machine OBJET et l'exécution de fonctions de mise au point ne doit pas modifier ce contexte (sauf si l'utilisateur le souhaite explicitement). Deux classes d'éléments matérialisent cet état :

* ceux qui sont directement accessibles dans l'unité centrale et qui ne doivent pas évoluer une fois que celle-ci est arrêtée. Nous voulons citer les registres généraux et à double précision, le mot d'état programme, les registres de contrôle.

* d'autres non accessibles, car ils sont rattachés à d'autres unités, mais qui influencent le fonctionnement de l'unité centrale. On peut citer : état des canaux d'entrée-sortie, état des unités de contrôle, état des unités d'entrée-sortie. Ils correspondent à des actions antérieures de l'unité centrale OBJET non encore terminées. C'est-à-dire que, dans l'exemple précédent, les interruptions asynchrones de fin d'opérations n'ont pas encore été générées.

Le fonctionnement de l'unité centrale ESPION est particulier : lorsque l'utilisateur demande l'activation de ce type de couplage, l'état de la machine OBJET est sauvegardé. Il ne doit être modifié que par des opérations initialisées directement par le système testé. C'est-à-dire que les fonctions de mise au point ne doivent pas altérer ce contexte. Cette condition est nécessaire si l'on ne veut pas que le système en cours de test ait une exécution différente de celle sans fonction de mise au point.

L'opération de couplage consiste donc essentiellement en une sauvegarde de l'état de l'unité centrale OBJET. La première classe d'éléments peut l'être immédiatement, tandis que la sauvegarde des éléments de la seconde classe est faite petit à petit au fur et à mesure que l'unité centrale ESPION travaille.

Toutes les conditions qui sont normalement réfléchies à l'unité centrale OBJET, sont données à celle de l'ESPION. Ceci permet un contrôle rigoureux de l'exécution.

La désactivation de cette relation d'inclusion entre unités centrales se fait en initialisant à nouveau l'unité centrale OBJET avec l'état tel que la machine ESPION l'a sauvegardé et fait évoluer au cours de l'exécution des opérations demandées par le système testé.

II-3-2-3 - Inclusion d'unités d'entrées-sorties

La machine ESPION dispose d'unités d'entrée-sortie pour communiquer au programmeur-utilisateur des résultats. Lorsque l'inclusion des unités centrales est effective, la machine ESPION dispose d'un jeu plus étendu de dispositifs d'entrée-sortie (les siens et ceux de la machine OBJET) et doit donc les faire fonctionner. Il est bien évident que la machine OBJET ne peut altérer l'état des unités propres à la machine ESPION.

Le seul problème soulevé par cette dernière partie de la relation d'inclusion, est provoqué par la nécessité de trier l'ensemble des interruptions asynchrones. En effet, certaines sont destinées aux fonctions de mise au point, d'autres sont à réfléchir à la machine OBJET immédiatement

et enfin d'autres, à donner à la machine OBJET, doivent être sauvegardées parce que cette dernière n'est pas en état de les traiter.

Il faut que l'ensemble des interruptions asynchrones liées aux dispositifs d'entrée-sortie soit envoyés à la machine ESPION.

I I-3-3 - Modes de fonctionnement du système SPY

Nous sommes maintenant capables de générer une machine ESPION et de la coupler à une autre machine virtuelle. Il faut exploiter cette nouvelle possibilité : telle est la seconde ambition du système de contrôle SPY.

Au paragraphe II.1 nous avons défini les exigences d'un système de contrôle. Que nous apporte la relation d'inclusion et quelle est son influence sur le fonctionnement du système de contrôle SPY ?

Le couplage des unités centrales permet de contrôler pas à pas le système résidant dans la machine OBJET. La relation d'inclusion entre ces deux ensembles entraîne la suppression temporaire de l'unité centrale OBJET. Dès lors l'unité centrale ESPION doit activer successivement deux classes de programmes : les aides à la mise au point et le système OBJET.

Pour être efficace le contrôle de l'exécution du système testé doit se faire par l'intermédiaire d'un simulateur d'instructions qui permet en particulier la mise en oeuvre d'outils dynamiques (voir paragraphe I.2.1). Ainsi l'unité centrale de la machine ESPION n'active plus qu'une seule classe de programmes :

- les outils de mise au point ou le simulateur pour exécuter les opérations du système OBJET.

L'état de l'unité centrale OBJET est fourni au simulateur lors de l'activation du couplage (voir §. II-3-2). Mais cet interpréteur doit disposer des informations contenues dans la mémoire de la machine OBJET, ainsi que des unités d'entrée-sortie de cette dernière. Donc l'inclusion des unités centrales entraîne l'inclusion des mémoires et l'inclusion des unités d'entrée-sortie.

Le couplage des mémoires donne la possibilité à une fonction propre à la machine ESPION d'examiner et de modifier le contenu de la mémoire de la machine OBJET. De par la nature de la relation d'inclusion, lorsque l'unité centrale ESPION exécute le système OBJET, une modification d'adresse est nécessaire. En effet, une instruction du système OBJET fait référence à l'élément d'adresse virtuelle r dans sa mémoire virtuelle. En fait, il veut utiliser l'élément d'adresse virtuelle k+r dans la mémoire de la machine ESPION. Il faut donc translater toutes les adresses générées par la machine OBJET ce qui n'est possible que grâce à une simulation.

Le couplage des unités d'entrée-sortie permet au système de contrôle de communiquer ses résultats et de tester le fonctionnement des composants assurant la gestion des unités de la machine OBJET. Ce type de couplage n'est utilisé que lorsque le couplage des unités centrales est actif.

Il ressort de ceci que les types de couplages sont intimement liés puisque l'activation de l'un nécessite généralement l'activation des autres. Mais aussi que le couplage en lui-même n'est pas d'une grande utilité si l'on ne dispose pas de programmes spécialement prévus pour l'exploiter. Le couplage de deux machines virtuelles conditionne la réalisation et le fonctionnement des composants de SPY. Quels sont les différents états du système de contrôle, et pour chacun quels sont les types de couplage qui sont activés ? (voir Fig. II.4).

Le système de contrôle SPY constitue un interface entre l'utilisateur et son système.

- Soit SPY dialogue avec l'utilisateur : ce dernier indique les opérations de mise au point (outils statiques) qu'il veut faire exécuter. Dans ce cas, l'unité centrale de la machine ESPION est active afin de prendre en compte ces opérations, l'unité centrale OBJET est arrêtée, en général les couplages mémoires et unités d'entrée-sortie sont actifs, afin de permettre un examen complet de la machine OBJET.

- Soit SPY exécute des opérations de mise au point (outils dynamiques) et les opérations initialisées par le système testé. Il est indispensable que tous les types d'inclusion soient actifs.

- Soit enfin, temporairement, l'utilisateur n'initialise plus d'outils de mise au point. La machine OBJET s'exécute de façon semi-autonome : l'unité centrale OBJET est active, mais toutes les conditions anormales (interruptions asynchrones) sont réfléchies à la machine ESPION. Provisoirement le couplage mémoire et unités d'entrée-sortie est désactivé.

II-3-3-1 - Mode_conversationnel

Chaque fois que le couplage des unités centrales est activé, ce mode de fonctionnement est disponible. L'unité centrale de l'OBJET a été arrêté et son état sauvegardé .

Ce mode permet à l'utilisateur, par l'intermédiaire d'un langage de requêtes, d'activer les divers outils de mise au point disponibles.

L'état de la machine OBJET ne doit pas être modifié puisqu'aucune des opérations de mise au point provoquent une exécution du système en cours de test.

Les requêtes tapées au terminal sont soit exécutées directement, soit rangées dans la mémoire libre de SPY. Cette dernière possibilité permet de mettre en place des fonctions de mise au point qui seront automatiquement activées lors de l'exécution du système testé. Naturellement ceci ne sera effectif que si les unités centrales sont incluses (Unité centrale de ESPION est la seule à pouvoir exécuter les opérations de mise au point).

Ce mode permet au programmeur d'utiliser les outils de mise au point disponibles dans le langage de requêtes de base ou s'il le désire, d'en créer de nouveaux : puisque nous disposons d'une machine entière pour établir notre panoplie d'aides, le mécanisme conversationnel d'extension du langage de commande donnera la possibilité d'écrire de véritables programmes de mise au point utilisables ensuite pour les tests.

MODE de fonctionnement de SPY

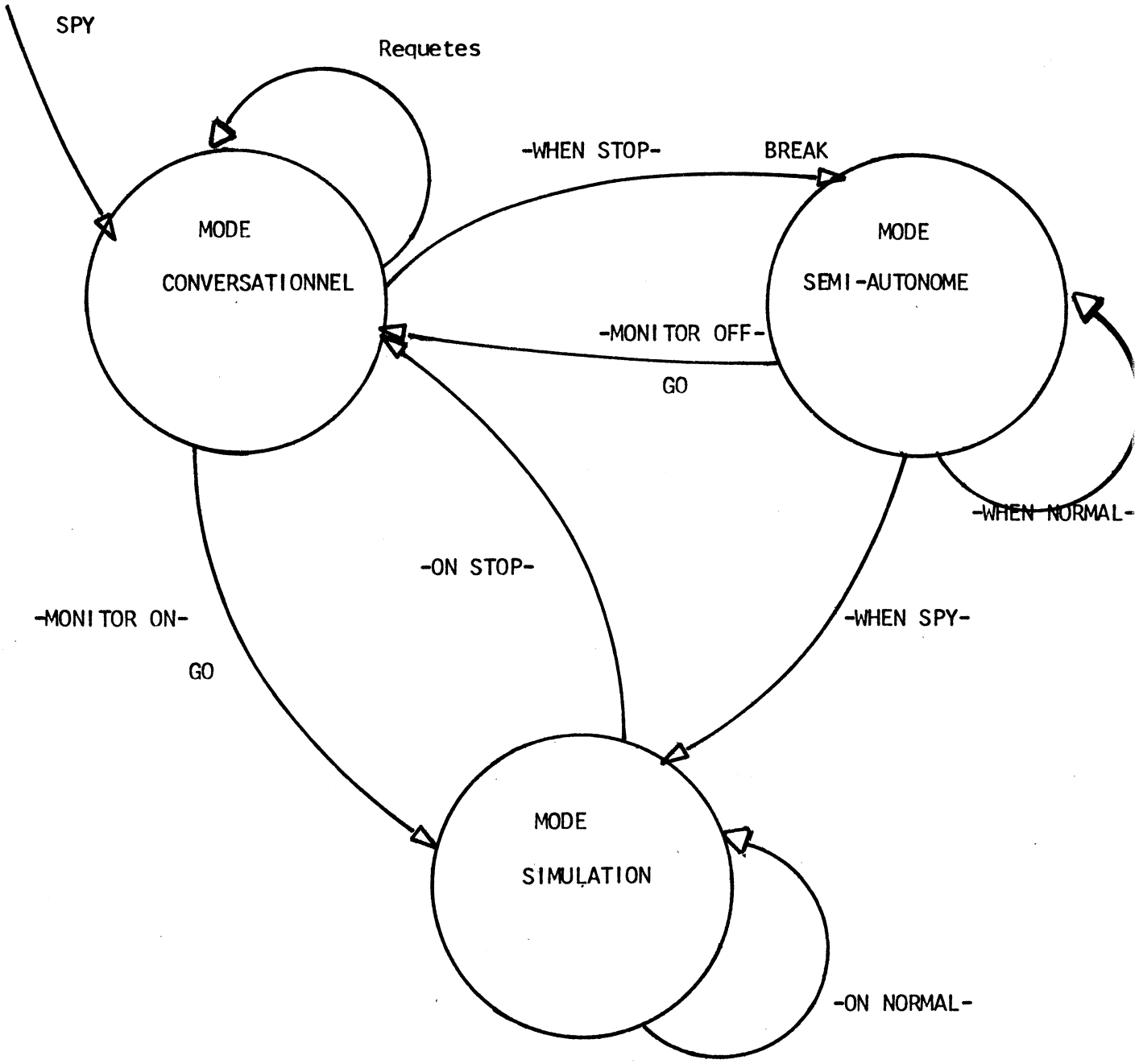


Fig. II.4

II-3-2-2 - Mode simulation

Toutes les formes du couplage sont actives : unité centrale, mémoire, unité d'entrée-sortie. L'unité centrale de la machine ESPION exécute les opérations initialisées par le système en cours de test et les fonctions de mise au point actives (outils dynamiques). Un simulateur d'instruction travaille dans la machine ESPION. Pour données, il utilise les instructions du système testé et l'état de la machine OBJET, et comme résultats il fournit les caractéristiques de l'instruction OBJET courante et une modification de l'état de machine OBJET équivalente à celle produite par l'exécution réelle de cette instruction. Les caractéristiques sont transmises aux opérations de mise au point actives. Le contrôle sur le système testé est total.

On passe du mode conversationnel à ce mode en activant le simulateur et en lui donnant le contrôle. Les outils de mise au point dynamique de SPY sont en fonction. Diverses sortes de points d'arrêt permettent de réactiver le mode conversationnel.

L'utilisateur peut choisir entre deux interpréteurs : l'un simule l'ensemble des dispositifs de l'unité centrale d'un 360 standard, l'autre les dispositifs de l'unité centrale d'un 360 modèle 67. Dans le second cas, il est possible de disposer de la translation dynamique d'adresse, du mot d'état programme étendu etc... Le tout étant simulé. Ces deux types d'interpréteurs sont décrits au chapitre III et aux références 25 et 26.

Ce mode ne doit être utilisé que lorsque l'on veut utiliser les outils de mise au point dynamique.

II-3-3-3 - Mode semi-autonome

Nous avons déjà énoncé que le mode simulation était très coûteux, en conséquence il convient de donner la possibilité de mettre hors fonction le couplage des unités centrales et par la même, celui des unités d'entrée-sortie et mémoires.

Chaque fois que le programmeur désire un contrôle plus souple, il doit désactiver temporairement le couplage entre les deux machines. Dès lors, puisqu'il ne désire plus initialiser de fonctions de mise au point, l'unité centrale de la machine OBJET peut exécuter normalement les instructions du système testé. Bien que n'étant plus active, la machine ESPION reçoit les conditions anormales (interruptions asynchrones) et remet en fonction le couplage.

Cette possibilité de travail n'est pas sans rappeler le fonctionnement classique d'un superviseur et d'un programme utilisateur. Ce dernier, lorsqu'il n'a pas besoin des services du superviseur, utilise l'unité centrale (le superviseur n'est pas actif). Mais lorsqu'une condition est anormale le superviseur est automatiquement réactivé et prend une décision quant à la suite des opérations.

Dans le cadre du mode de fonctionnement semi-autonome, le schéma est identique. Bien que non actif, le système de contrôle SPY reçoit les interruptions asynchrones, ce qui provoque l'arrêt de la machine OBJET, l'activation du couplage des deux machines, et en fonction de directives données décide du mode de fonctionnement à initialiser.

On peut classer les conditions anormales en deux groupes :

- Interruptions asynchrones :
  Elles provoquent une rupture dans l'ordre séquentiel d'exécution. En général un système de programmation arrête le programme qui était actif pour effectuer le traitement correspondant à l'interruption.

  Leur importance est d'autant plus grande que la plupart des systèmes ont des politiques de travail dirigées par ces interruptions. Il est donc primordial de permettre au programmeur de faire un choix quant à la suite des opérations lorsque ces interruptions apparaissent.

  Certaines sont souhaitées telles les opérations de fin d'opérations d'entrée-sortie ou les interruptions d'appel au superviseur, d'autres sont néfastes, telles les interruptions pour faute de programmation. D'autres encore permettent d'arrêter une exécution anarchique telle

interruption dite "externe" qui permet de toujours réactiver SPY.

- Les instructions privilégiées :
  Grâce à elles un système de programmation peut modifier l'état de l'unité centrale et même plus généralement l'état du calculateur. On peut ainsi changer le mot d'état programme ou faire démarrer une opération d'entrée-sortie etc... Dans tous les cas il est bon que l'utilisateur définissent la conduite à tenir.

Cette méthode de travail que nous appelons mode semi-autonome présente de grands avantages par rapport au mode totalement autonome (le système testé s'exécute normalement sans possibilité d'intervention du système de contrôle) ou par rapport au mode simulation. En particulier, elle est beaucoup moins coûteuse et beaucoup moins longue tout en permettant un contrôle suffisamment précis. Un système de programmation se distingue d'un programme "banal" par le fait qu'il est capable de traiter les interruptions asynchrones et de modifier l'état du calculateur par l'exécution d'instructions privilégiées. Conserver une trace de ces événements exceptionnels, permettre un changement de politique lors de leurs apparitions semble grandement souhaitable. Quant aux instructions banales, elles seront exécutées librement tout en conservant la possibilité de placer quelques points d'arrêt ne nécessitant pas la simulation.

En fait ce mode de travail, très souple, est le plus utilisé ; le mode simulation n'étant activé que dans des cas très rares.

Nous avons défini six conditions permettant de réactiver le couplage : les cinq types d'interruptions asynchrones (fin d'opération d'entrée-sortie, appel du superviseur, externe, erreur machine, erreur de programmation) et l'exécution d'une instruction privilégiée.

Pour tenir compte de cette dernière condition, nous obligeons le système en cours de test à travailler en mode "esclave" (exécution d'une telle instruction dans ce mode provoque une interruption) et les instructions privilégiées sont toujours simulées par SPY.

La rencontre de l'une de ces six conditions provoquent une réactivation automatique du couplage et SPY décide quelle est la conduite à tenir. Un ensemble de quatre directives permet pour chaque condition d'imposer à SPY un mode de travail :

- NORMAL La condition est ignorée, le système OBJET continue à s'exécuter librement. C'est toujours le mode semi-autonome qui est actif. Cette directive ne peut être combinée avec d'autres.

- TRACE Un message est envoyé à l'utilisateur lui indiquant la condition apparue, et les modifications à l'état du calculateur qui ont été faites. Le mode de travail est indiqué par l'une des deux directives suivantes.

- SPY Le mode simulation est automatiquement activé. Les points d'arrêts les paramètres pour la trace des instructions qui auraient été précisées auparavant restent valables.

- STOP Le mode conversationnel est mis en fonction. L'utilisateur peut taper à son terminal des requêtes. Il peut en particulier choisir entre le mode simulation et le mode semi-autonome.

Ces directives ont été choisies pour prévenir à toutes circonstances accidentelles du programme testé, et permettre au programmeur d'agir avant qu'une erreur ne vienne perturber l'exécution et compliquer la mise au point. Ainsi, l'utilisateur peut exercer un contrôle précis sur le fonctionnement de sa machine et choisir son mode de travail de façon à ce que le temps d'exécution ne devienne pas prohibitif.

II-3-4 - Outils de mise au point de SPY

Les aides à la mise au point nécessaires à un travail efficace ont été défini par B. PETEUL-HARMEL (voir référence 26). Ils sont été développés dans PILOTE. Nous avons repris le même ensemble dans SPY avec certaines modifications.

Nous distinguons dans SPY deux classes outils : les outils statiques activables depuis le mode conversationnel et les outils dynamiques qui sont utilisés lors de l'exécution du système testé. Il faut aussi citer deux fonctions particulières : les points d'arrêt ne nécessitant pas l'activation du simulateur et la fonction de définition du mode semi-autonome (elle permet de préciser pour chaque condition les directives souhaitées). Certains "outils de confort" comme l'adressage symbolique ont été abandonné. Ils obligent à une trop grande connaissance du système testé qui doit avoir été assemblé dans des conditions particulières, charger en mémoire suivant une méthode particulière etc... Le mécanisme d'extension de PILOTE a été abandonné. Nous pensons le remplacer par un autre système d'extension basée sur l'utilisation de CMS qui permettrait d'écrire et de réaliser de nouveaux outils de mise au point assimilable directement dans le langage de requêtes de SPY.

Cet ensemble de fonctions réside dans la mémoire propre à la machine ESPION et utilise ces propres unités d'entrée-sortie. Il est totalement protégé des instructions anarchiques du système en cours de test par les propriétés de la relation d'inclusion.

II-4 - CONCLUSION

Le système de contrôle SPY est formé d'un ensemble de fonctions implantées dans le système CP-67 et d'un ensemble d'aides à la mise au point directement inspiré de celui de PILOTE.

Le premier ensemble définit un environnement favorable à la mise au point, en particulier les programmes du système SPY sont parfaitement protégés contre toutes erreurs du système en cours de test et ce dernier ignore les contrôles dont il fait l'objet.

Le second fournit à l'utilisateur des fonctions de mise au point permettant soit un examen complet de sa machine virtuelle, soit un contrôle rigoureux de l'exécution du système testé, soit un contrôle rigoureux uniquement aux phases critiques. Il dispose pour ce faire d'un ensemble de programmes utilitaires tels que : gestion de la mémoire libre, gestion des interruptions asynchrones, etc...

Aucune modification ne doit être apportée et au système de contrôle et au système à tester.

CHAPITRE - III

DESCRIPTION LOGIQUE DU SYSTEME SPY

Le système de contrôle SPY se compose de deux ensembles de fonctions très différentes. Le premier génère une machine virtuelle ESPION, le second fournit les outils de mise au point adaptés à l'environnement particulier dont nous disposons.

La construction d'une machine ESPION, c'est-à-dire la réalisation de la relation d'inclusion entre deux calculateurs, ne peut se faire qu'au niveau de la définition de ces machines virtuelles. Si nous voulions mettre en application les propriétés que nous venons de définir sur deux ordinateurs réels, nous serions obligés de créer de nouvelles liaisons électroniques. Notre but, moins ambitieux, est de les mettre en oeuvre sur des machines virtuelles ; ce qui nous amène à définir de nouvelles liaisons entre machines virtuelles. Ceci est fait en ajoutant au système CP-67 de nouvelles propriétés et les programmes permettant de coupler deux machines virtuelles. La représentation des machines virtuelles a été modifiée, de nouvelles propriétés utilisables avec un objectif autre que celui du système SPY ont été développées.

Pour construire les outils de mise au point du système SPY nous avons repris la totalité des travaux de B. PETEUL. Nous avons implanté dans notre environnement particulier les fonctions du support PILOTE et nous avons étendu ses possibilités.

Par exemple les moyens de contrôle lorsque le système testé s'exécute librement ont été améliorés et un nouveau simulateur d'instruction a été réalisé. Il permet d'interpréter des programmes utilisant les dispositifs technologiques propres aux ordinateurs de la série 360 type 67.

III-1 - GENERATION D'UNE MACHINE VIRTUELLE ESPION

Une machine ESPION possède la propriété fondamentale de pouvoir augmenter dynamiquement le nombre de ses ressources et cela au détriment d'une autre machine virtuelle que nous avons nommée machine OBJET (paragraphe II.3.1.). Ceci permet de définir une relation d'inclusion entre ces deux calculateurs virtuels : la machine ESPION peut utiliser, donc contrôler, les ressources de la machine OBJET, cette dernière ne pouvant procéder à l'opération inverse.

Le système CP-67 simule un certain nombre de composants électroniques utilisables par un programme. Ainsi les registres généraux ou flottants, le mot d'état programme sont représentés par des quantités programmées. Des dispositifs originaux tels que l'échange des mots d'état programme lors de l'apparition d'une interruption sont représentés par des fonctions du système CP-67.

Nous avons introduit un niveau supplémentaire entre le programme testé et le calculateur réel : les fonctions du système SPY. En effet, ce dernier met à sa disposition du système OBJET un jeu d'instructions plus grand, des registres généraux simulés, un mot d'état programme simulé. C'est aussi le système SPY qui dirige la substitution des mots d'états programmes citée précédemment.

La possibilité de créer un niveau supplémentaire dans le système CP-67 a été pour nous un grand avantage puisque nous utilisons les mécanismes de protection de ce système. En particulier il nous assure un rempart contre toute volonté de destruction de la part du système testé.

Nous pouvons affirmer que la mise en fonction du couplage consiste à dresser une liste des ressources possédées par la machine ESPION et par la machine OBJET et à la communiquer au système CP-67. Quant à la désactivation nous pouvons la résumer en un retrait de la liste des ressources de la machine ESPION de l'ensemble des listes connues du système CP-67.

Il est très important que l'opération de générer une machine ESPION se fasse tout en conservant l'état exact des diverses quantités manipulées : que se soit l'unité centrale, les unités d'entrée-sortie ou la mémoire ; aucune des informations représentants ces éléments ne doit être perdue, oubliée ou modifiée.

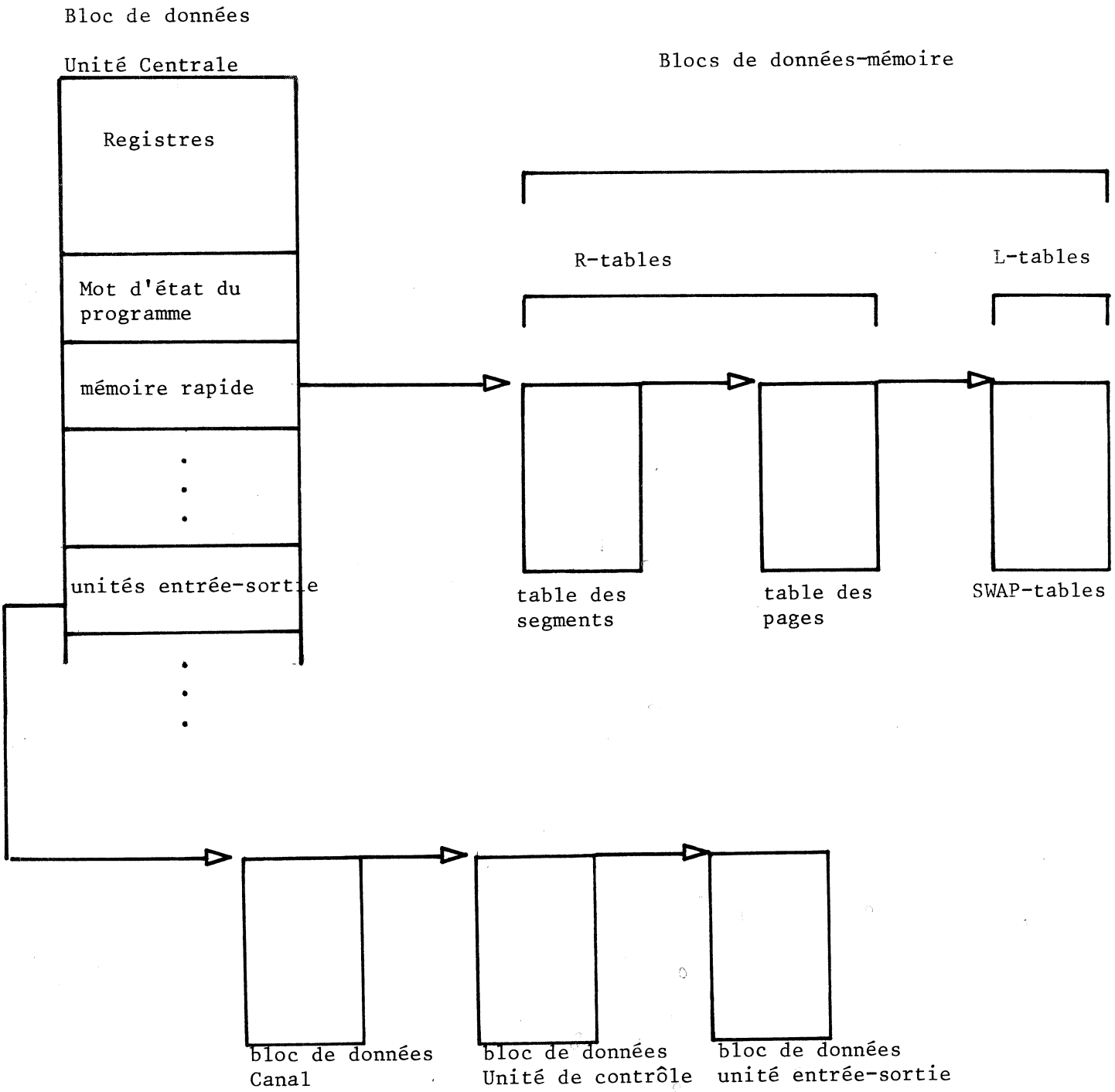


Fig. IV.1
Représentation d'une machine virtuelle par le système CP-67.

III-1-1 - INCLUSION DES UNITES CENTRALES

III-1-1-1 - Description d'une unité centrale virtuelle

La simulation, par le système CP-67, de cette partie d'un ordinateur est obtenue en affectant périodiquement l'unité centrale du calculateur réel à la machine virtuelle. Ceci implique que deux programmes se partagent cet élément réel : le système CP-67 lui-même et le système de programmation résidant dans la machine virtuelle (référence 15, 24, Fig. IV.1).

Chaque fois que l'unité centrale réelle n'est plus affectée au calculateur virtuel il faut sauvegarder les éléments de cette unité qui ont été modifiés : registres généraux et flottants, le mot d'état programme, l'horloge, les interruptions asynchrones non encore traitées et certains registres spéciaux dans le cas où la machine virtuelle est un ordinateur 360 du type 67. Les contenus correspondants sont rangés dans un bloc mémoire. Ce bloc représente ainsi à chaque instant l'état de l'unité centrale virtuelle. Ces données reprendront place dans leurs homologues réels lors d'une nouvelle affectation de l'unité réelle à ce calculateur virtuel.

Ainsi une machine virtuelle dispose d'un jeu d'instructions au moins identique à celui de la machine réelle. Néanmoins, les instructions dites "privilégiées" susceptibles d'altérer CP-67 ont un traitement particulier : en effet le système CP-67intercepte ces instructions et simule leur fonctionnement afin de se protéger contre toute faute de programmation.

Le bloc de mémoire appelé UTABLE, contient les éléments de l'unité centrale qui sont modifiés par l'exécution du système de programmation résidant dans la machine virtuelle correspondante. On y trouve essentiellement :

* les registres généraux et à double précision.
* le mot d'état programme.
* le pointeur initial vers les tables décrivant la mémoire virtuelle.
* la taille de la mémoire virtuelle.
* les pointeurs initiaux vers la description des dispositifs d'entrée-sortie.

* les interruptions asynchrones en attente de traitement (entrée-sortie, externe).

* le fonctionnement de la machine virtuelle
  - en attente et pour quelle raison
  - en cours de création
  - en attente de fonction console etc...

* l'horloge du calculateur virtuel.

Un certain nombre de renseignements complémentaires sont indispensables pour matérialiser l'état d'une machine virtuelle en particulier un certain nombre de pointeurs repèrent cette UTABLE. Ils permettent de communiquer à la machine de nouvelles conditions de fonctionnement. Par exemple, lorsqu'une entrée-sortie est initialisée il faut en garder trace afin de pouvoir réfléchir à la machine virtuelle émettrice l'interruption de fin d'opération.

III-1-1-2 - Description de l'inclusion

La fonction essentielle de l'inclusion est de permettre à l'utilisateur de disposer d'un jeu plus étendu d'opérations élémentaires. Ces nouvelles possibilités autorisent l'exécution des fonctions de mise au point et le contrôle du déroulement du système OBJET.

La réalisation du couplage des unités centrales est guidée par une contrainte fondamentale :

- il faut préserver l'état de la machine OBJET et l'exécution des fonctions de mise au point ne doit pas modifier ce contexte.

Nous avons dit précédemment que l'état du calculateur virtuel est matérialisé par deux classes d'éléments, l'une étant directement accessible et l'autre non. Cette dernière classe est indirectement rattachée à l'unité centrale virtuelle (à la UTABLE) par l'intermédiaire de pointeurs.

0	NSPYUTAB	SYYUTAB suivante
4	SPVGPR	registres généraux
44	SPVFPR	registres flottants
64	SPVPSW	mot d'état programme
6C	SPSEGTAB	adresse de la table des segments
70	SPVMASIZ	taille de la mémoire virtuelle
74	SPUSYSTA	système partagé
78	SPSEGTABD	déplacement table des segments depuis début du bloc mémoire libre.
7C	OLDUNIT	adresse table des unités
80	MPXSPY	adresse unités sur le canal multiplex
84	SPYCONSO	adresse description du terminal
88	SPYARG	argument de la fonction console SPY
8C	SPYCON	adresse du terminal
90	SPYCHAN6	adresse description du canal d'adresse 6
94	OLDUNISI	nombre d'unités virtuelles

Fig. IV.2 - Description de la SPYUTAB "attachée" à la UTABLE de la machine OBJET.

Dès lors il devient évident que la UTABLE de la machine virtuelle OBJET doit représenter l'état de l'unité centrale active du couple ESPION-OBJET (la machine OBJET peut fonctionner avant d'initialiser le couplage, c'est-à-dire qu'il existe dans CP-67 des pointeurs permettant de retrouver ce bloc mémoire).

Puisque l'unité centrale qui est active doit recevoir toutes les interruptions y compris celles provenant d'ordres émis dans un autre mode de fonctionnement, il nous a semblé plus facile d'attribuer tour à tour cette table à l'une ou l'autre des deux machines :

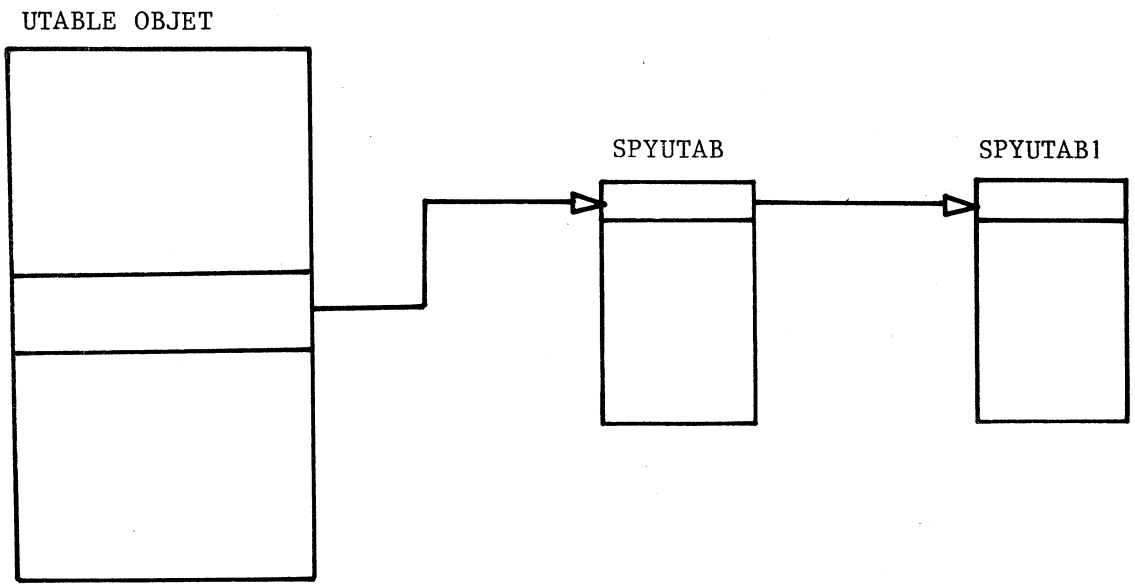
* lorsque la machine OBJET est une machine virtuelle quelconque, aucun couplage n'ayant été initialisé, la UTABLE de cette machine reflète son état.

* lorsque le couplage machine ESPION-machine OBJET est actif, l'utilisateur manipule les fonctions de mise au point, la "UTABLE OBJET" représente l'état de l'unité centrale ESPION. Le contenu antérieur de cette table sert de données pour les programmes de mise au point et pour le simulateur d'instructions.

* lorsque le couplage est temporairement désactivé, l'utilisateur ne souhaite plus disposer des fonctions de mise au point tout en définissant des conditions anormales qui réactiveront le couplage, la UTABLE représente l'état de la machine OBJET. Les quantités définissant l'état de la machine ESPION sont sauvegardées dans une extension de la UTABLE. Elles appartiendront de nouveau à cette table lors de l'apparition d'une condition anormale.

Donc chacune des deux unités centrales est représentée par un ensemble de données. Cet ensemble réside ou non dans la UTABLE de l'OBJET suivant l'unité centrale qui est active.

Quant aux conditions extérieures (interruptions asynchrones par exemple) elles sont toujours réfléchies à la machine virtuelle ESPION. Si celle-ci occupe la UTABLE cette réflexion est normale. Par contre si cette unité centrale n'est pas



SPY1 contrôle	SPY qui contrôle	OBJET
↓	↓	↓
UTABLE Objet	SPYUTAB1	SPYUTAB
SPY1 désactivé	SPY qui contrôle	OBJET
↓	↓	↓
SPYUTAB1	UTABLE Objet	SPYUTAB
SPY1 désactivé	SPY désactivé	Objet s'exécute normalement
↓	↓	↓
SPYUTAB1	SPYUTAB	UTABLE Objet

Fig. IV.3

utilisée, c'est le système CP-67 lui-même qui prend l'initiative de réactiver le couplage et le contrôle est donné à un programme spécial du système SPY.

Cette procédure est équivalente à la "procédure hardware" de permutation des mots d'état programme sur un ordinateur 360, puisque l'apparition d'une interruption asynchrone provoque l'activation d'un programme de traitement. Dans notre cas ceci s'accompagne d'une autre fonction : réactiver le couplage des deux machines.

L'unité centrale qui n'est pas active est matérialisée par une extension de la UTABLE OBJET appelée SPYUTAB (Fig. IV.2). Activation et désactivation du couplage consiste en une permutation des informations contenues dans ces deux tables. Les informations contenues dans la SPYUTAB ne sont manipulées que par les fonctions du système CP-67 liées aux opérations de couplage.

Ainsi la protection et la sauvegarde des quantités placées dans la SPYUTAB est automatiquement assurée : ni le système CP-67 (fonctions classiques autres que celles du système SPY), ni à plus forte raison le système OBJET ne connaissent l'existence de ces informations. Toute altération ou destruction est de par ce fait impossible.

L'itération de ce processus est simple, il suffit de créer de nouveaux blocs (SPYUTAB) qui sont chaînés aux précédents. Le dernier bloc représente toujours la machine OBJET ou la machine ESPION la plus récemment créée (Fig. IV.3).

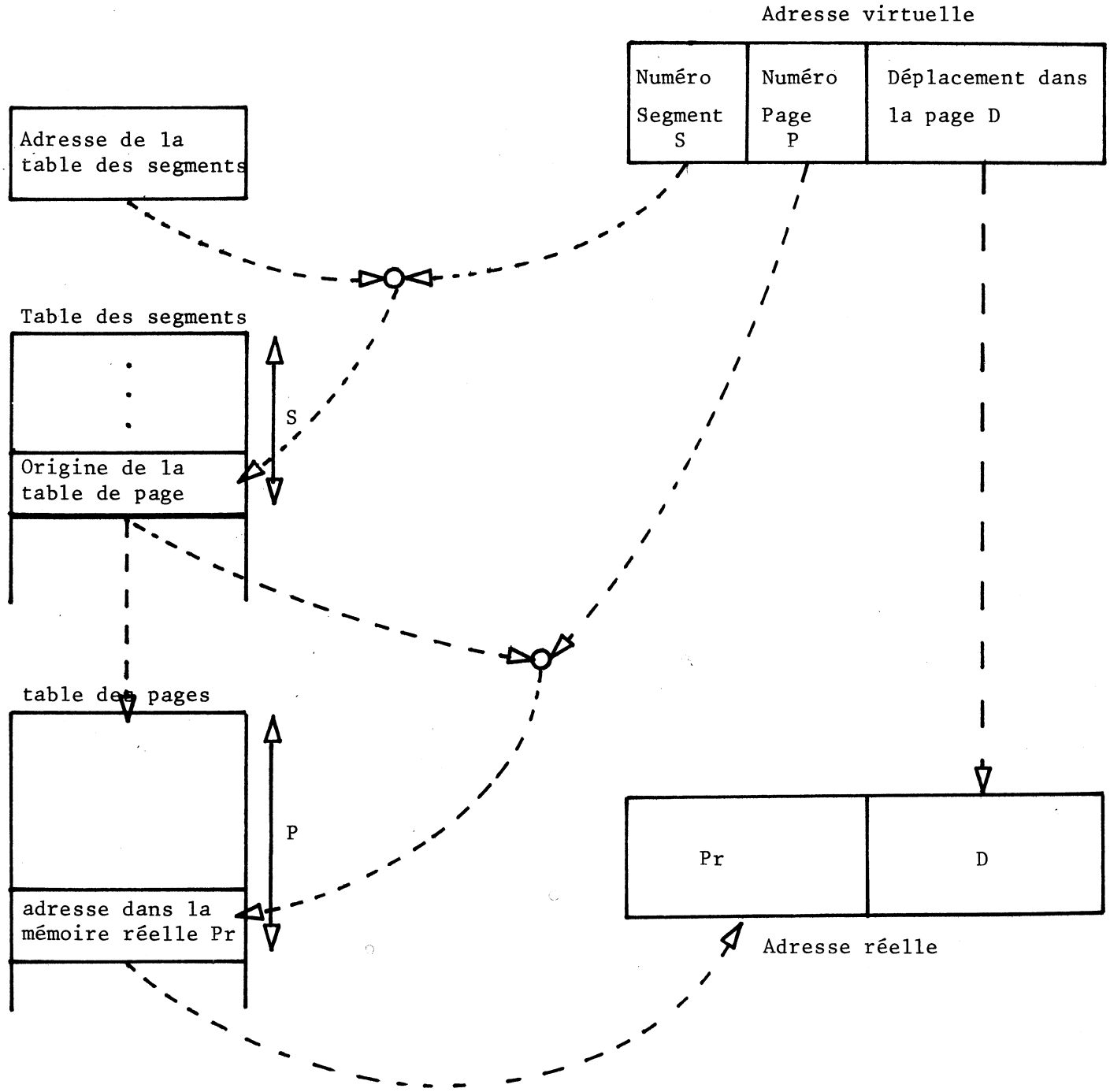


Fig. IV.4 Translation dynamique d'adresse : Réalisation pratique

III-1-2 - INCLUSION DES MEMOIRES VIRTUELLES

III-1-2-1 - Description de la mémoire d'une machine virtuelle

Nous avons décrit au paragraphe II.3.2.1. le principe d'une mémoire virtuelle. Décrivons maintenant une réalisation de ce principe, c'est-à-dire comment le système CP-67 génère pour chaque machine virtuelle une mémoire rapide.

Une mémoire virtuelle est divisée en blocs de longueur fixe appelé pages (une page = 4096 octets). Elles sont groupées par ensemble de 256 pages consécutives nommé segment.

Une mémoire virtuelle est représentée par deux familles de tables : les R-tables et les L-tables. Les R-tables permettent de retrouver les pages de la mémoire virtuelle qui résident dans la mémoire réelle, les L-tables quant à elles permettent de retrouver celles qui résident en mémoire secondaire. Dans la terminologie propre à l'IBM 360/67 (référence 16) les R-tables s'appelent tables des segments et tables des pages et les L-tables, dans la terminologie du système CP-67 se nomment les SWAP-tables (référence 15).

Un dispositif électronique de traduction dynamique d'adresse (D.A.T.), convertit les adresses générées par un programme résidant dans la mémoire virtuelle en leur homologue dans la mémoire réelle. Pour établir cette correspondance, ce dispositif consulte les tables citées précédemment (table de segment et table de page Fig. IV.4).

Si l'adresse résultante est une adresse dans la mémoire secondaire alors la page désignée doit être amenée en mémoire centrale et l'adresse réelle recalculée. Les SWAP-tables permettent d'effectuer ces transferts de page entre mémoire secondaire et mémoire réelle et vice-versa. Ce sont des fonctions du système CP-67 qui déterminent le lieu de résidence d'une page d'une mémoire virtuelle et pour ce faire elles seront amenées à modifier les tables décrivant la mémoire virtuelle. Ces fonctions ne doivent pas permettre que deux mémoires virtuelles se recouvrent, ou en d'autre terme,

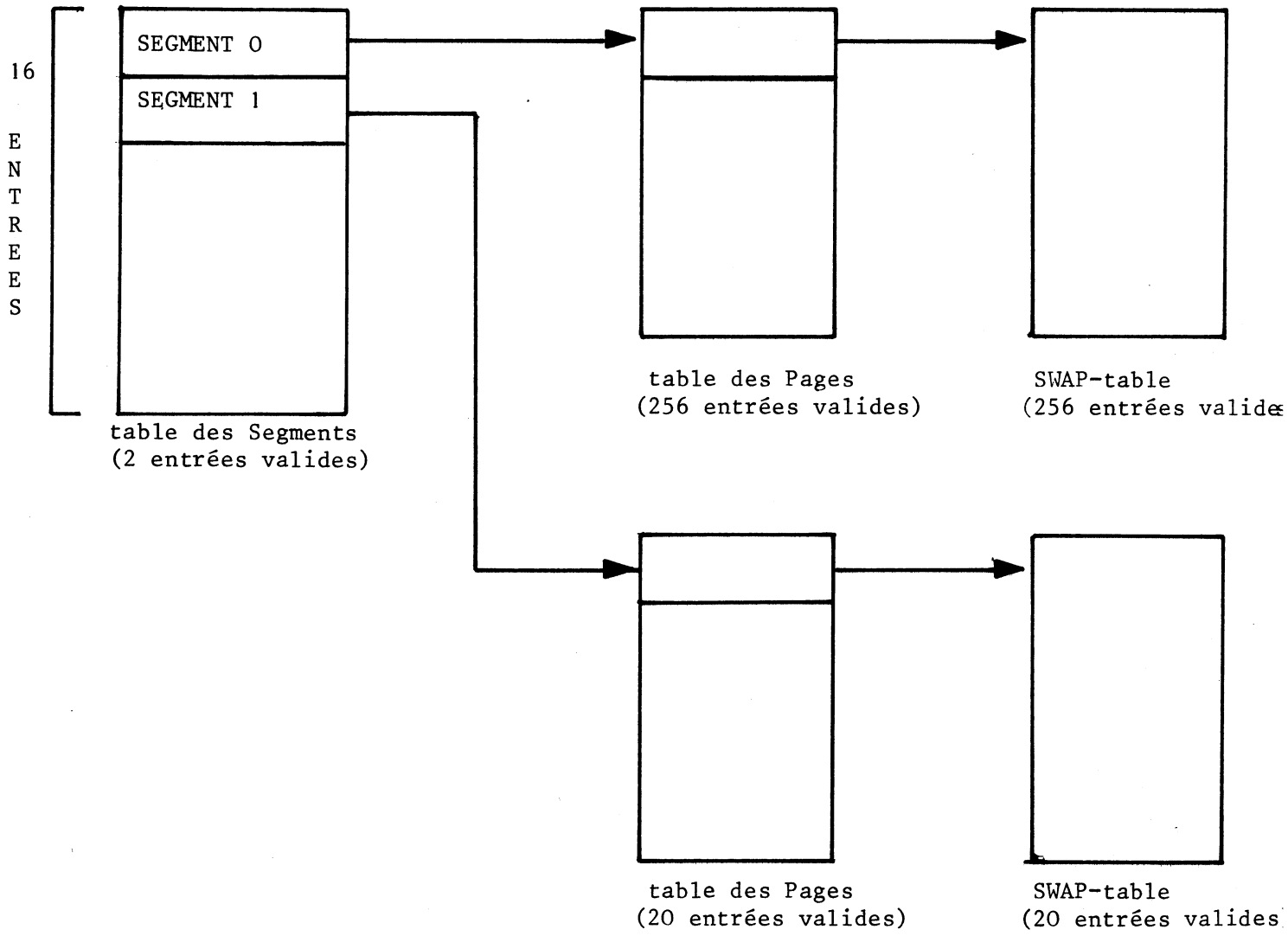


Fig. IV.5 - Description d'une mémoire virtuelle de 276 pages (1104 K octets).

doivent interdire à un programme résidant dans une mémoire virtuelle de référencer une quantité ne se trouvant pas dans la même mémoire.

Pour cela il suffit que les tables décrivant la mémoire virtuelle ne donnent pas accès à d'autres informations que celles qui sont contenues dans celle-ci. Le dispositif technologique (D.A.T.) assure cette protection puisque toute adresse virtuelle (générée par un programme résidant dans la mémoire virtuelle) lui est soumise.

La table des segments comprend 16 entrées dont, le plus souvent, seules les premières ont un contenu valide. Ainsi une mémoire virtuelle comportant au plus 256 pages est représentée par une seule entrée dans la table des segments, une table de pages et une SWAP-table. Si par contre, la mémoire virtuelle est formée d'un nombre de pages entre 256 et 512 alors deux entrées dans la table des segments sont utilisées et à chacune d'elles est associée une table de pages et une SWAP-table (Fig. IV.5)

Habituellement, sous CP-67 lorsqu'une mémoire virtuelle est multi-segment seul le dernier peut être décrit par une table de pages ayant moins de 256 entrées.

III-1-2-2 - Description de l'inclusion

Au chapitre II nous avons défini la relation d'inclusion entre mémoires virtuelles. La mémoire de la machine ESPION est divisée en deux parties :

* l'une contient les programmes propres au système SPY.
  Ils permettent de tester le système OBJET - MVS.
* l'autre est une copie de la mémoire de la machine OBJET - MVO.

Un certain nombre de remarques vont nous premettre de définir ces deux parties.

Une instruction résidant dans la partie MVO ne doit pas pouvoir référencer une quantité se trouvant dans MVS. Les adresses générées par une telle instruction sont des quantités positives, mais qui dans le cas d'une instruction anarchique peut être une valeur supérieure à celles autorisées par la taille de la mémoire virtuelle.

Si l'on définit l'ensemble P = {0,1,...,N} où 0,1,...,N sont les numéros des pages de la mémoire ESPION, nous affecterons les pages de numéro 0,1,...,k à la partie MVS et les pages k+1,...,N à la partie MVO. Donc dans notre relation d'inclusion l'ordre est primordial.

Ce découpage présente un autre avantage. Dans l'architecture de l'ordinateur 360 les mots de mémoires dont l'adresse est comprise entre 0 et 256 constituent une zone privilégiée. Divers éléments de l'unité centrale y rangent des informations qui sont ensuite utilisées par le système de programmation actif sur cette machine. En particulier on y trouve des renseignements rangés lors de l'apparition d'une interruption asynchrone, et le mécanisme de permutation des mots d'états programmes sur interruption trouve là les données nécessaires à son fonctionnement. Puisque la machine ESPION doit recevoir l'ensemble des interruptions destinées au couple de machines il est primordial que cette zone privilégiée lui appartienne. Ce qui est vrai avec notre découpage.

D'après la description que nous venons de donner d'une mémoire virtuelle nous voyons qu'elle est découpée en blocs totalement autonomes : les segments. Chaque bloc est décrit par un ensemble de tables, la jonction des segments n'étant assurée qu'au niveau de la table des segments. Nous avons affecté au système SPY le segment 0, et la mémoire de la machine OBJET est recopiée dans les autres segments (1,...,15).

Lorsque le couplage est actif, c'est-à-dire lorsque le couplage unité centrale et le couplage mémoire sont actifs,la mémoire virtuelle utilisable comprend N+1 segments (N+1 toujours inférieur ou égal à 16). Lorsque le couplage est temporairement désactivé la mémoire virtuelle utilisable ne comporte plus que N segments.

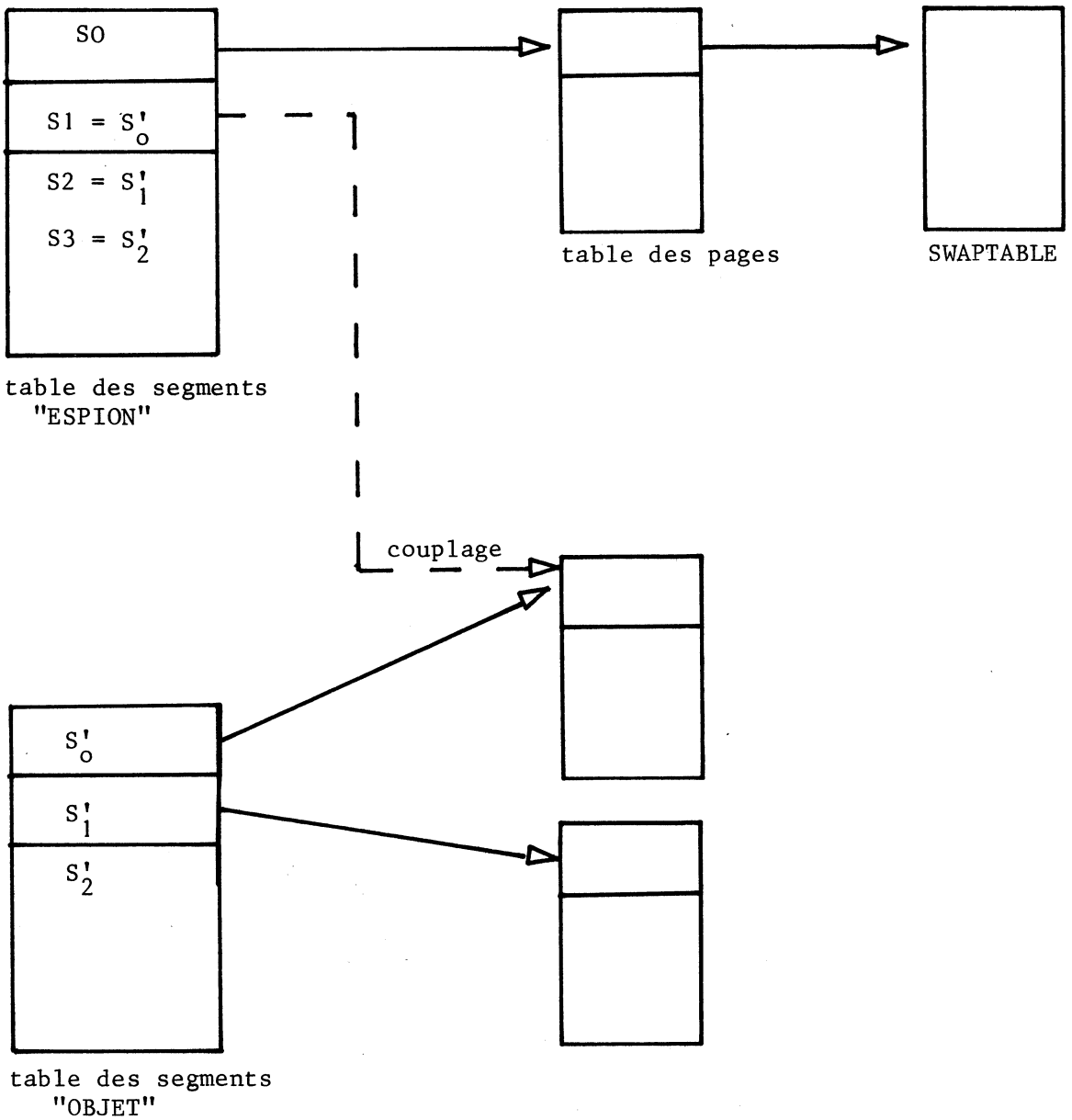


Fig. IV.6 - Inclusion de mémoires virtuelles.

Le processus d'activation et de désactivation consiste en une simple manipulation de la table des segments. Le segment du système SPY y est inséré lorsque le mode de fonctionnement est soit le mode conversationnel, soit le mode simulation. Ce même segment est retiré lorsque le mode semi-autonome est actif. Cette insertion provoque un décalage d'une entrée pour les autres segments (Fig.IV.6).

Cette technique permet de conserver de façon simple les modifications apportées lors des fonctionnements précédents puisque les opérations de couplage se limitent en une manipulation de la représentation de la mémoire virtuelle en non en une manipulation des éléments de cette mémoire. Lorsque le mode semi-autonome est actif la protection des outils de mise au point est totale puisque l'entrée correspondante dans la table des segments a disparu.

Lorsque le couplage mémoire est actif et que le système testé s'exécute une simulation des instructions est nécessaire. En effet, les adresses générées par le système en cours de test font référence à une mémoire virtuelle constituée des segments 0,1,... Maintenant ce système fait toujours référence aux mêmes segments de la mémoire OBJET qui sont en fait les segments 1,2,... de la mémoire ESPION. La simulation a donc pour fonction entre autre d'augmenter toutes les adresses générées par le système OBJET de 100.000 en hexadécimal (ceci décale les adresses de un segment).

Par contre en mode semi-autonome, le segment 0 n'étant plus utilisé, aucune fonction de mise au point n'est initialisée. Cette simulation n'est plus nécessaire.

Cette façon de procéder peut paraître une contrainte mais en fait elle assure une protection totale des programmes propres au SPY. Si le segment de ces programmes ne se trouve pas dans la table des segments, il n'est pas utilisable et son contenu ne peut être modifié. Si le segment est présent dans la table, alors le système OBJET s'exécute sous le contrôle du simulateur qui interdira toute référence à cet emplacement de la mémoire ESPION.

Cette méthode de couplage nous a conduit, par soucis d'économie, à développer une nouvelle propriété pour les machines virtuelles. Celles-ci peuvent avoir une mémoire virtuelle répartie sur plusieurs segments non complets (moins de 256 pages). En effet, il est inutile de disposer d'un segment complet (1024 K octets) pour implanter les fonctions du système SPY, donc une grande partie de la table de pages et de la SWAP-table attachées à ce segment sont inutilisés.

Puisque les segments 1,2,... sont utilisés ces tables doivent comporter 256 entrées. Nous avons supprimé cette contrainte, évitant ainsi d'immobiliser inutilement de la place en mémoire réelle.

III-1-3 - INCLUSION DES UNITES D'ENTREE-SORTIE

III-1-3-1 - Description des unités d'entrée-sortie virtuelles

Dans l'architecture des ordinateurs 360 les dispositifs périphériques d'entrée-sortie sont regroupés en trois classes : les canaux, les unités de contrôle et les unités proprement dites. Comme il a été dit auparavant une machine virtuelle possède une configuration différente de la configuration réelle. Il faut, dès lors, définir des liens entre unités réelles et unités virtuelles, modifier les programmes canaux "virtuels" pour les adapter aux unités réelles.

Chaque élément de l'ensemble des dispositifs d'entrée-sortie est représenté par un bloc mémoire relié d'une part au blocs décrivant d'autres éléments et d'autre part aux blocs représentant les unités réelles. Celà permet de garder et de connaître l'état d'une unité virtuelle et de la projeter sur l'unité réelle qui la simule.

III-1-3-2 - Description de l'inclusion

Cette partie du couplage consiste à relier les tables décrivant les unités de la machine ESPION aux tables de la machine OBJET lorsque le couplage est actif et de les détacher dans le cas contraire.

Les unités de la machine ESPION sont de deux types :

* les unités de canal multiplexeur : imprimante, lecteur-perforateur de cartes, terminal etc... Une machine virtuelle ne peut posséder qu'un seul canal de ce type ; aussi les unités de la machine OBJET et de la machine ESPION y sont rattachées.

* les unités "attachées" à des canaux simples : les disques par exemple. Une machine virtuelle peut posséder jusqu'à six canaux de ce type. Nous avons donc décider de lier au canal d'adresse 6 toutes les unités de cette sorte, propres à la machine ESPION. En conséquence, la machine OBJET ne doit pas attacher de dispositifs d'entrée-sortie sur ce canal 6. L'expérience a montré que ce n'est pas une contrainte.

La difficulté de l'inclusion apparaît au niveau de l'utilisation contrôlée des unités de la machine OBJET. Nous avons dit que les deux machines ESPION et OBJET ont des unités liées à un même canal et plus généralement la même console opérateur (voir paragraphe IV.1) est utilisée par les deux systèmes correspondants. Nous avons établi deux conventions :

* Lorsque la machine ESPION veut utiliser pour son propre compte une unité d'entrée-sortie elle doit faire en sorte que cette opération soit transparente pour l'OBJET. C'est-à-dire que SPY doit remettre le canal, l'unité de contrôle, l'unité proprement dite dans leur état initial.

* La machine ESPION n'utilise pas la simultanéité lorsqu'elle émet des ordres d'entrée-sortie. C'est-à-dire qu'elle attend l'interruption indiquant que l'opération est terminée.

Pour que cette dernière condition soit réalisée, le système SPY doit autoriser la prise en compte de toutes les interruptions arrivant sur le canal qu'il utilise. Dès lors la machine ESPION peut recevoir des informations destinées à la machine OBJET alors qu'elle n'est pas en mesure de les lui envoyer. Il faut donc préserver les informations adéquates telles que le mot d'état du canal. Lorsque le système SPY a terminé son travail il doit remettre l'unité dans l'état où il l'avait trouvé. Malheureusement le système SPY n'est pas maître du canal et les phénomènes étant liés au temps, il est

impossible de les reproduire exactement. Le système SPY envoie une opération spéciale sur l'unité périphérique concernée, cette "pseudo-opération" a pour seul but d'occuper à nouveau le canal et de générer une interruption.

Lorsque le mot d'état programme du système OBJET permet de recevoir des interruptions asynchrones sur le canal particulier, le système OBJET est interrompu par la réflexion par le système SPY de l'interruption de fin de la pseudo-interruption. Il suffit qu'un programme spécialisé de SPY ait restauré les informations sauvegardées précédemment (mot d'état du canal etc...) pour que le système OBJET ait l'illusion de traiter la véritable interruption de fin d'opération.

Ce procédé, quelque peu compliqué, justifie l'utilisation du canal d'adresse 6 par la machine ESPION seule : dans ce cas particulier cette démarche est inutile.

III-1-4 - ACTIVATION ET DESACTIVATION DU COUPLAGE

L'ensemble des fonctions que nous venons de décrire permettent de coupler deux machines virtuelles et autorise l'une à contrôler l'autre. Nous avons développé et inclus dans le système CP-67 une importante notion : le couplage actif et le couplage temporairement désactivé sont réalisé par un mécanisme de bascule ; de cette manière, soit le mode simulation, soit le mode semi-autonome est disponible ce qui conduit à une grande souplesse de fonctionnement. Décrivons tout d'abord la fonction console SPY.

III-1-4-1 - La fonction console SPY

Elle transforme la machine virtuelle désignée en une machine OBJET, crée éventuellement une machine ESPION et réalise le couplage entre les deux machines et finalement active le mode conversationnel du système SPY. La machine OBJET peut être désignée implicitement (c'est la machine OBJET qui a émis la fonction console SPY) auquel cas il y a création d'une machine ESPION. Elle peut aussi être désignée explicitement : depuis la machine

ESPION on émet la fonction console SPY suivie du nom de la machine OBJET (voir § IV.1).

Cette commande a donc pour but d'initialiser le processus de couplage. Pour ce faire elle doit :

* créer l'extension de la UTABLE de la machine OBJET et l'initialiser avec les quantités qui décrivent l'état de cette machine.

* installer dans la UTABLE les informations représentant l'unité centrale de la machine ESPION.

* recopier dans la table des segments de la machine ESPION celle de l'OBJET à partir du segment 1 (le segment 0 est réservé au système SPY).

* mettre en place la représentation des unités d'entrée-sortie propres à la machine ESPION. Nous ne nous occupons que du canal multiplexeur, c'est-à-dire de l'imprimante, du lecteur-perforateur et du terminal. Si la fonction console a été émise depuis une machine ESPION il faut rattacher à cette description le terminal de cette machine de façon à disposer de deux terminaux (voir paragraphe IV.1). Le canal d'adresse 6 n'est initialisé que dynamiquement. Lorsque l'utilisateur a émis la commande SPY et que la machine ESPION est active, il peut attacher des unités sur ce canal (au moyen de la fonction console LINK voir référence 14). Cette procédure permet à plusieurs utilisateurs de demander simultanément l'aide du système de contrôle SPY sans pour cela provoquer de conflits au niveau de ces unités.

* initialiser le segment 0, c'est-à-dire permettre en place les programmes qui permettent de tester le système OBJET.

* activer le mode conversationnel du système SPY.

III-1-4-2 - Passage du mode simulation au mode semi-autonome

Le passage du mode simulation au mode semi-autonome et vice-versa se réalise grâce aux requêtes "MONITOR OFF" et "MONITOR ON" (voir § IV.2.3.).

La désactivation du couplage est effectuée en réactivant les paramètres décrivant la machine OBJET. Le segment 0 propre à la machine ESPION disparait la table des segments représente à nouveau la mémoire virtuelle de l'OBJET. Les unités de la machine ESPION sont enlevées. L'unité centrale de la machine ESPION n'étant plus appelée à travailler temporairement, les paramètres qui la décrivent dans la UTABLE de la machine OBJET sont remplacés par ceux décrivant l'unité centrale de la machine OBJET. Ce dernier ensemble de paramètres est communiqué par les fonctions du système SPY (ils auront été modifiés par une éventuelle exécution du système testé). Les quantités relatives à l'état de l'unité centrale de la machine ESPION sont sauvegardées dans la seconde table (SPYUTAB). Seul demeure dans la UTABLE de la machine OBJET un indicateur signalant que le couplage est temporairement désactivé.

Par la suite, chaque fois que le système OBJET veut exécuter une instruction privilégiée, chaque fois que le système CP-67 veut réfléchir à la machine OBJET une interruption asynchrone, le couplage est automatiquement rétabli. C'est le système SPY qui, dès lors, doit déterminer le mode de fonctionnement ultérieur : faut-il ou non désactivé à nouveau le couplage ? Si le couplage reste actif lequel des deux modes simulation ou conversationnel doit être activé ? Ces différentes possibilités ont été fixées par la requête WHEN (voir § IV.2.3.) qui a été émise avant l'activation du mode semi-autonome.

L'activation du couplage est une opération simple : l'état de la machine OBJET est sauvegardé dans la SPYUTAB, puis communiqué au système SPY, l'état de la machine ESPION est implanté dans la UTABLE de la machine OBJET, le segment 0 est inséré, les unités d'entrée-sortie insérées.

III-2 - DESCRIPTION DU SEGMENT 0 DE LA MACHINE ESPION

Cette partie de la mémoire virtuelle de la machine ESPION contient tous les programmes nécessaires à la mise au point. Ils comprennent un simulateur d'instructions, un analyseur de requêtes, des modules de gestion de la mémoire et des entrée-sortie de la machine ESPION. Cet espace virtuel n'est adressable que lorsque la machine ESPION est active, c'est-à-dire lorsque le mode de fonctionnement est soit le mode simulation soit le mode conversationnel.

III-2-1 - Initialisation du segment

Nous verrons au chapitre IV que la machine ESPION est généralement créée au moment où l'utilisateur demande l'aide du système de mise au point. C'est-à-dire que la création de cette machine et son couplage à un autre calculateur virtuel ont lieu simultanément. Une troisième action doit accompagner les deux précédentes : l'initialisation de la mémoire propre à la machine ESPION et l'activation des programmes qui y sont implantés.

Pour réaliser cette opération il faut amener dans le segment 0 de l'ESPION, le système de mise au point et l'ensemble des fonctions de base de ce système.

Une séquence de chargement, de tous les composants, entraîne l'exécution d'un nombre important d'entrée-sortie commandées par un programme spécial du système SPY appelé "chargeur".

Examinons les problèmes posés par cette procédure : si l'opération de mise en activité du système SPY est en cours, c'est qu'une machine virtuelle a demandé les services de celui-ci ; de ce fait l'état instantané de cette machine virtuelle a été figé et en particulier toutes les opérations d'entrée-sortie en cours sont restées en suspend. On peut dire alors que toutes les interruptions destinées à la machine virtuelle sont mises en attente, ceci implique que toute autre opération d'entrée-sortie ne peut être autorisée. Or le programme "chargeur" doit être averti de l'achèvement de ses propres opérations d'entrée-sortie. On constate ainsi que la séquence normale de chargement initiale (encore appelée IPL sur les ordinateurs 360) ne peut être utilisée.

Pour faire face à une telle situation, nous avons utilisé une technique particulière au système CP-67 appelé "IPL par nom". Donnons quelques détails liés à cette particularité :

Considérons un système quelconque qui doit prendre place dans une mémoire de machine virtuelle. Un tel système possède évidemment plusieurs séquences d'initialisation indispensables à son fonctionnement. Ces séquences sont exécutées après que le système ait été implanté dans la mémoire.

Le mécanisme fournit par CP-67 conduit à copier en mémoire secondaire le dit système après que toute sa phase d'initialisation soit réalisée ; en conséquence, l'aspect global est sauvegardé (valeur du mot d'état programme, partie de la mémoire virtuelle occupée) et une SWAP-table est construite (paragraphe III.1.2) et également sauvée.

Le système CP-67 connait donc avec précision l'emplacement de cette image instantanée du système prêt à fonctionner ; il peut réaliser lui-même l'opération de chargement quand celle-ci est demandée. Ceci revient à recopier dans la SWAP-table de la mémoire virtuelle de la machine intéressée, la SWAP-table du système sauvegardé.

Cette façon de procéder ne nécessite pas l'exécution d'entrée-sortie, et notre machine ESPION peut ainsi être mise en activité sans entraîner de perturbations au niveau du système testé qui reste dans l'état précis où il était quand il a demandé l'aide du système SPY.

III-2-2 - CONSTITUANTS DU SEGMENT

Les composants du système SPY qui résident dans la machine ESPION ont un triple but :

* mettre à la disposition des utilisateurs des fonctions de mise au point.
* permettre à ces fonctions de communiquer avec le monde extérieur.
* créer un mécanisme automatique d'extension du jeu d'outils de base (cet option est en cours de réalisation).

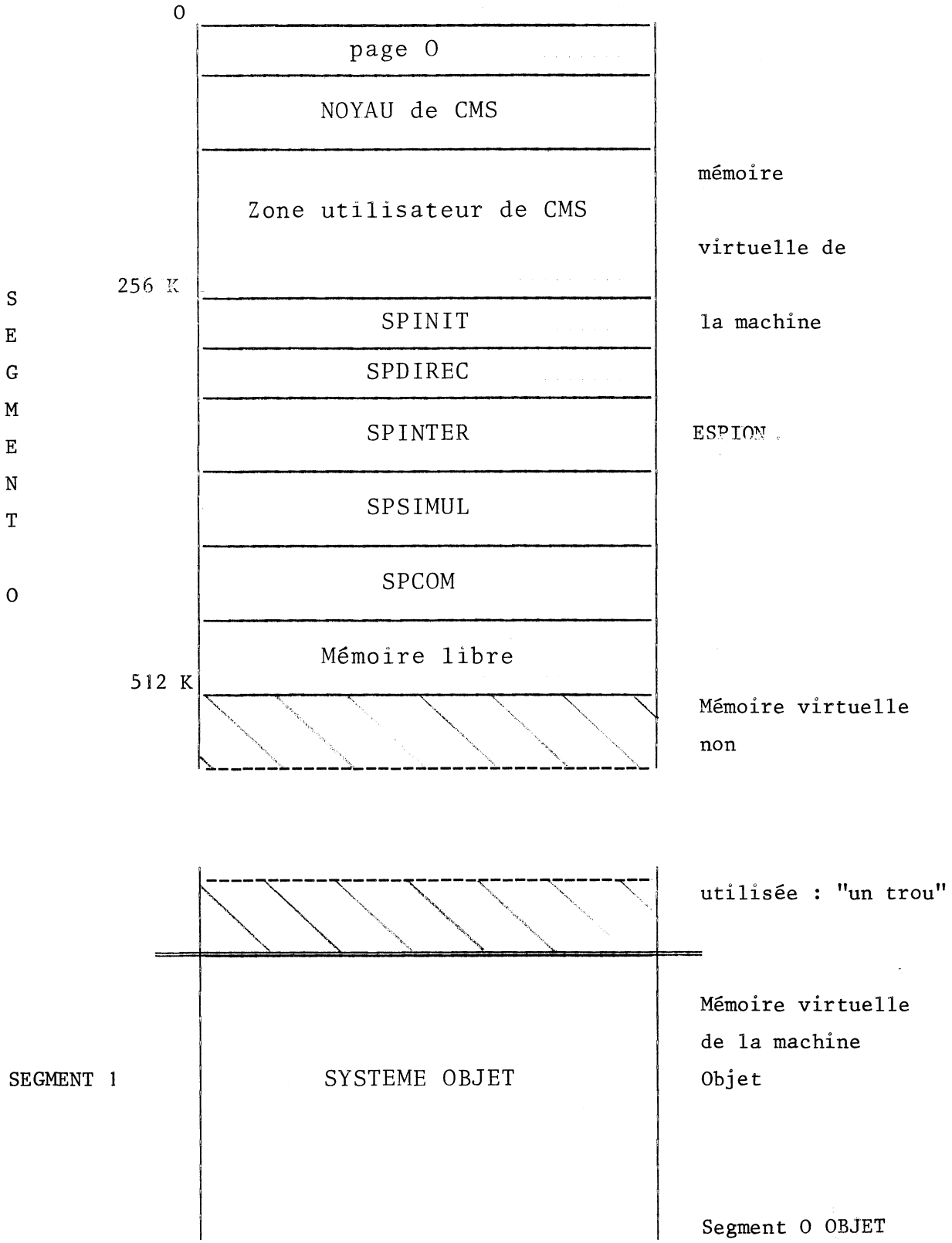


Fig. III.7 - Description du segment 0 de la machine ESPION.

Pour cela nous avons divisé le segment 0 en deux parties totalement indépendantes (figure III.7).

La première constituée des emplacements mémoires d'adresse 0 à 40.000 (hexadécimal) reçoit un sous-système conversationnel. Nous avons choisi le système CAMBRIDGE MONITOR SYSTEM (CMS) qui possède déjà un éditeur de fichiers, un ensemble restreint d'outils de mise au point, un assembleur et des compilateurs. Grâce aux diverses possibilités de ce sous-système nous pourrons développer, augmenter et améliorer la mise au point de façon à rendre opérationnel un large éventail d'outils pour satisfaire les demandes les plus exigeantes. Nous pourrons aussi adapter le macro-langage propre à CMS (EXEC) de telle sorte qu'il soit possible de combiner les outils de base de la mise au point pour définir de nouvelles fonctions plus élaborées dont la nécessité est apparue dans de nombreux cas particuliers.

L'ensemble de ces possibilités sera disponible dans le mode conversationnel qui possède un programme de reconnaissance et d'analyse automatique de toutes les requêtes.

Les fonctions de mise au point sont implantées dans la seconde partie de ce segment. On peut les classer en cinq ensembles.

- SPINIT

  Ce programme initialise les composants de la machine ESPION lors de la première activation du couplage. En particulier il initialise le système CMS de façon à ce que ce dernier utilise la zone de mémoire virtuelle comprise entre 0 et 256 K octets. Il active l'environnement conversationnel du système SPY à qui il transmet les paramètres qui matérialisent l'état de la machine OBJET.

- SPDIREC

  Ce programme gère toutes les entrées-sorties des fonctions de mise au point Il lui incombe la tâche délicate d'émettre les opérations d'entrée-sortie du mode conversationnel, de la trace etc... sans modifier l'état des unités de la machine OBJET. Une fois cet ordre émis, SPDIREC attend l'interruption de fin d'opération et replace, éventuellement, dans leur état initial, les unités de la machine OBJET qui auraient pu être affectées.

- SPINTER

  Ce composant est activé avec le mode semi-autonome. C'est à lui que le système CP-67 réfléchit toutes les interruptions destinées à l'OBJET. Dès lors suivant les directives de la requête WHEN précédemment émise il décide du mode ultérieur de fonctionnement.

- SPSIMUL

  C'est l'unité centrale simulée. Actuellement cette unité est celle d'un ordinateur 360 standard.

III-3 - LE SIMULATEUR D'INSTRUCTIONS

La plus importante fonction du système de contrôle SPY est le contrôle dynamique de l'exécution, ce qui autorise la mise au point de programmes sans qu'aucune modification ne soit apportée à ceux-ci.

Cette fonction est rendue possible grâce à un simulateur (ou interpréteur) d'instructions qu'il faudrait, en fait, appeler un simulateur d'unité centrale.

L'interpréteur simule le fonctionnement complet de l'unité centrale d'un ordinateur IBM 360. Il crée un environnement de simulation qui est invisible du programme testé. Une exception doit être faite pour le temps d'exécution qui est plusieurs dizaines de fois supérieur à celui d'une exécution non simulée. Il doit donc traiter deux entités les interruptions asynchrones et les instructions.

L'interpréteur utilise l'unité centrale donc, toutes quantités de cette dernière utilisables par un programme doivent être simulées pour le programme soumis à l'interpréteur. Il crée des pseudo-registres contenant les valeurs des registres généraux correspondant au programme testé. De la même façon un pseudo-mot d'état programme est utilisé pour indiquer l'instruction courante du programme interprété et pour indiquer l'état de l'unité centrale simulée.

L'interpréteur a accès à l'instruction courante et lui fait subir une phase d'analyse. Ceci lui donne la valeur de certains paramètres tels que : code opération, registres, les emplacements mémoires utilisés etc...

Ces renseignements sont utilisés pour construire une nouvelle instruction produisant un résultat identique mais mettant en jeu les pseudo-registres, le pseudo-mot d'état programme... Après l'interprétation de l'instruction courante le pseudo-mot d'état programme est mis à jour de telle sorte qu'il indique la prochaine instruction à exécuter. Cette opération est très importante dans le cas où l'instruction simulée est une rupture de séquence.

L'interpréteur fait peu de contrôle sur la validité des instructions à simuler. Son fonctionnement dépend des erreurs qui se trouvent dans le programme testé. Ceci implique qu'il soit capable de traiter ces erreurs et surtout de les réfléchir correctement au programme simulé. Il est donc nécessaire de garder dans les pseudo-quantités les valeurs exactes correspondant à l'instruction courante.

La simulation de l'unité centrale ne peut être complète que si l'interpréteur rend également compte des interruptions. Pour celà il ne doit prendre en con que celles qui sont autorisées par le programme testé afin de ne pas avoir à fair face à de délicats problèmes de rangement des informations fournies et de restitution au programme interprété.

A un tel simulateur d'unité centrale peuvent être couplés des programmes construisant une trace ou un historique de l'exécution, vérifiant si certaines zones mémoires ne sontpas référencées, mettant en place des points d'arrêts permanents etc...

Actuellement dans le système SPY un simulateur d'ordinateur IBM 360 standard est opérationnel.

Puisqu'une machine virtuelle dans le système CP-67 peut être un calculateur IBM 360 type 67, il nous a semblé important de développer un simulateur pour ce type d'unité centrale. Si les principes fondamentaux restent les mêmes dans le deux interpréteurs, il faut tenir compte des nouveaux dispositifs électroniques propres à l'IBM 360/67. En particulier le format du mot d'état programme est modifié, de nouveaux registres -les registres de contrôle- sont introduits, le mécanisme de traduction automatique d'une adresse virtuelle en une adresse réelle est implanté, de nouvelles instructions sont disponibles. Une option fondamentale a été prise : le simulateur fonctionne sur une unité centrale d'un IBM 360 standard Nous pensons que ceci est nécessaire afin de permettre l'utilisation de ce programme dans un cadre autre que celui du système SPY.

La réalisation de ce composant a été confiée à deux étudiants du D.E.A. (référence 25). Elle se décompose en plusieurs étapes :

- Etudier et programmer une simulation des dispositifs originaux du 360/67.

- Construire un simulateur d'instructions ne mettant pas en jeu le type de l'ordinateur 360.

- Réaliser un mécanisme permettant d'utiliser soit une unité centrale simulée standard soit une unité centrale simulée du type 67. Le changement de l'une à l'autre pouvant être produit par des instructions du programme simulé.

Il doit permettre de tester des systèmes tels que CP-67. La mise en fonction de cet interpréteur dans le système SPY amènera une modification sensible des commandes disponibles afin de pouvoir utiliser les nouvelles possibilités qu'il offre. En particulier il sera nécessaire de pouvoir préciser si les adresses données en paramètre dans les commandes sont des adresses virtuelles ou des adresses réelles

CHAPITRE - IV

MANUEL DE L'UTILISATEUR DU SYSTEME SPY

L'utilisation du système de contrôle SPY peut se découper en trois grandes phases :

- Création d'une machine ESPION et réalisation du couplage avec la machine virtuelle intéressée, laquelle devient une machine OBJET

- Utilisation des outils de base de mise au point, construction de nouveaux outils adaptés à l'activité envisagée.

- Abandon du système SPY.

Nous examinerons successivement ces trois phases en indiquant pour chacune d'elles les commandes ou les requêtes disponibles.

IV-1 - ACTIVATION DE SPY

Après avoir demandé la création de sa machine virtuelle depuis un terminal, un utilisateur peut mettre en fonction un système de programmation de son choix à l'aide de la fonction console IMP (simulant la fonction "hardware" de chargement initial). La machine virtuelle est alors dirigée par ce système de programmation et le terminal devient la console opérateur pour le système (référence 14, 15).

Comme nous l'avons énoncé précédemment (paragraphe I.1) l'utilisateur peut remettre son terminal dans l'état "pupitre" de sa machine virtuelle à l'aide de la touche "ATTN". Dans cet état, il dispose des fonctions consoles définies au paragraphe I.1 (un résultat identique est obtenu avec la fonction console SET ADSTOP décrite au paragraphe I.1).

L'une de ces fonctions consoles permet de demander la création d'une machine ESPION et la réalisation du couplage à la machine virtuelle émettrice. C'est la fonction SPY.

L'ensemble des opérations effectuées par cette fonction SPY peut se résumer ainsi :

* mise en activité de la machine virtuelle ESPION

* Transformation de la machine émettrice en machine OBJET, c'est-à-dire réalisation de ce que nous avons appelé l'inclusion.

* Initialisation du mode conversationnel.

En conséquence les segments 1 à 15 de la mémoire de l'ESPION sont réservés à l'OBJET

le segment 0 de cette mémoire recueille le système SPY et les outils de mise au point (voir chapitre II et III).

Cette façon d'initialiser le système SPY présente l'avantage d'utiliser qu'un seul terminal. En effet SPY et le système de programmation testé se partagent la même console opérateur. Ce partage qui doit respecter l'indépendance du système en cours de mise au point, est effectué par un composant de SPY . Le terminal est utilisé par le mode conversationnel, par la trace et aussi par le système OBJET.

Nous avons dit précédemment, que dans ce cas le terminal est partagé par les deux machines. Le système SPY traite les conditions d'erreurs sur cette unité sans les réfléchir au système testé. Cette façon de procéder peut ne pas convenir au programmeur qui souhaite mettre au point les fonctions de contrôle du terminal. Une seconde procédure d'activation est disponible qui autorise l'utilisation de deux consoles : l'une appartenant au système SPY et l'autre appartenant au système testé.

Dans ce cas, la machine ESPION et la machine OBJET effectuent séparément leur procédure d'initialisation. Par la suite, depuis le terminal de la machine ESPION, la fonction console SPY XXX conduit à prendre en test la machine dont le nom est donné en paramètre de la fonction console (ici XXX) et à effectuer les mêmes opérations que nous avons citées ci-dessus.

Cette procédure soulève le problème du droit d'utiliser le système SPY. En effet, la première méthode permet à l'utilisateur de choisir le moment où il désire activer le système SPY et les informations sur son système en cours de test seront connues de lui seul. Par contre, la seconde méthode donne la possibilité à un utilisateur quelconque "d'espionner" le travail d'un autre utilisateur et celà à son insu. Un dispositif de protection est mis en place. La seconde procédure ne peut être activée que depuis une machine virtuelle particulière qui n'est connue que des gens habilités à utiliser ainsi le système SPY . Cette méthode est efficace puisqu'il faut connaître la clé permettant de créer cette machine particulière (création d'une machine virtuelle, référence 14).

Fonction console SPY

Format :

SPY <OBJET>

OBJET désigne le nom de la machine virtuelle qui est transformée en machine OBJET . Ce nom est implicite si l'on travaille avec un seul terminal (activation du système SPY depuis la machine de l'utilisateur.

Utilisation :

Cette fonction console établit le couplage entre la machine ESPION et la machine OBJET. Elle initialise aussi l'ensemble des outils de mise au point et active le mode conversationnel.

Réponses :

1 - SPY IS READY
Le système SPY est correctement initialisé. Le mode conversationnel est actif. Il est matérialisé par le signe - qui apparaît en tête de chaque ligne.

2 - CP ARG 01
Le système de la machine OBJET est un "système partagé" (au sens défini dans CP-67). SPY ne peut être initialisé. Aucune fonction de couplage n'a été réalisée.

3 - SYSTEM VOLUME XXXXXX NOT MOUNTED
Le disque contenant le système SPY n'est pas connu du système CP-67.

4 - MEMORY SIZE TOO LARGE
La taille mémoire de la machine OBJET possède déjà 16 segments. Elle ne peut être coupler à la mémoire de la machine ESPION . Aucune fonction de couplage n'a été réalisée.

Remarques :

1 - La machine ESPION créée par la première émission de la fonction console SPY peut à son tour devenir une machine OBJET ; il suffit d'émettre depuis le terminal une seconde fois la fonction console SPY . Chaque nouvel appel insère un niveau supplémentaire de couplage. Cette procédure est utile pour mettre au point une version expérimentale du système SPY.

2 - La fonction console SPY suivie d'un argument ne peut être émise que depuis une machine particulière. (machine virtuelle SPY). Si le nom donné ne correspond à aucune machine active le message :
USER NOT FOUND
s'imprime au terminal.

```
 login archives
ENTER PASSWORD:
*********
READY AT 08.13.12 ON 05/24/72
VP
ipl 190
CMS..VERSION 3.0  janvier 72

 VP
query virtual
 CORE - 00768K
 009 CONSL 050
 010 SPOOL RDR
 00C SPOOL RDR
 00D SPOOL PUN
 020 SPOOL PRT
 00E SPOOL PRT
 0FF RPQ TIMER
 190 2314- 290 054 CYL R/O VCPD1
 191 2314- 290 010 CYL R/W VCPD1
 290 2314- 290 203 CYL R/W VCPD1
 380 2314- 290 003 CYL R/O VCPD1

 display psw
PSW = FF060000 00002306

 spy
 -SPY READY
_psw
FF06000000002306
_VP
query virtual
 SEGMENT  00 00512K
 SEGMENT  01 00768K
 009 CONSL 050
 010 SPOOL RDR
 00C SPOOL RDR
 00D SPOOL PUN
 020 SPOOL PRT
 00E SPOOL PRT
 0FF RPQ TIMER
 0EC SPOOL RDR
 0ED SPOOL PUN
 0EE SPOOL PRT
 190 2314- 290 054 CYL R/O VCPD1
 191 2314- 290 010 CYL R/W VCPD1
 290 2314- 290 203 CYL R/W VCPD1
 380 2314- 290 003 CYL R/O VCPD1
```

Définition de la configuration de la machine OBJET

demande de création de la machine ESPION

Segment réservé à SPY
Segment reservé au système OBJET

Nouvelles unités propres à la machine ESPION

IV-2 - UTILISATION DU SYSTEME SPY

Après avoir demandé la construction du couple machine ESPION machine OBJET , l'initialisation de SPY et l'activation initiale du mode conversationnel, le travail de mise au point peut devenir effectif.

Dans ce mode conversationnel, l'utilisateur peut examiner le calculateur OBJET (valeur du mot d'état programme, valeur des registres généraux, contenu des mots mémoires etc,...), il peut aussi modifier certaines valeurs. L'utilisateur choisi aussi le mode de travail qui lui convient le mieux :

- mode simulation grâce à la requête MONITOR ON.

- mode semi-autonome grâce à la requête MONITOR OFF.
  Dans ce cas il doit préciser pour les six conditions particulières les directives qu'il souhaite (voir paragraphe II.3.3.3.) C'est la requête WHEN.

- abandon définitif du couplage grâce à la requête LOGSPY.
  La machine ESPION est détruite, la machine OBJET redevient une machine virtuelle quelconque.

L'utilisateur peut maintenant commander l'exécution contrôlée du système OBJET s'il existe (c'est-à-dire s'il a été chargé dans la mémoire de la machine OBJET avant le couplage) grâce à la requête GØ ; ou bien il peut demander un chargement contrôlé de son système grâce à la requête IPLSPY.

En effet, il peut être nécessaire de surveiller la première phase de l'initialisation d'un système : chargement en mémoire centrale des composants du système testé. La requête IPLSPY simule la fonction "hardware" de chargement initial et construit à l'adresse 0 de la mémoire OBJET un mot d'état programme qui permet d'initialiser le processus programmé de chargement. Dès lors l'initialisation de la mémoire OBJET avec les composants du système à tester, peut se faire sous le contrôle des aides à la mise au point.

Nous avons dit au paragraphe II.3.4. que la panoplie de base des outils de mise au point est pratiquement identique à celle fournie par
aussi nous avons gardé la même formulation du langage de commandes. Cette solution présente, en particulier, l'avantage d'offrir à l'utilisateur un jeu identique d'outils de mise au point au cours des deux phases distinctes de la mise au point :

- test des modules pris séparément.

- test de l'ensemble des modules.

Nous donnons ici un classement des requêtes de ce langage accompagné d'une description succincte. Les spécifications détaillées de chacune d'elles sont mentionnées dans la référence 26.

IV-2-1 - Entrées dans le mode conversationnel

Il est activé lors de l'initialisation du système SPY , puis à chaque apparition d'une condition anormale.

Ce mode de fonctionnement est disponible dans les deux cas suivants :

- le mode de travail est le mode simulation.
  Le simulateur arrête l'exécution du système testé et imprime sur le terminal la cause de cet arrêt.
  Celui-ci est provoqué :

  * Soit par un arrêt sur référence à une zone de mémoire indiquée auparavant dans la requête STOP REFERENCE . L'arrêt a lieu avant l'exécution de l'instruction qui fait référence à cet emplacement.

  * Soit par la rencontre d'une instruction indiquée auparavant dans la requête STOP INSTRUCTION . Comme précédemment l'exécution de cette instruction n'a pas encore eu lieu.

  * Soit sur l'occurrence d'une interruption d'un certain type, laquelle a été précisée par la requête ON.

* Soit sur la rencontre d'un point d'arrêt permanent qui a été spécifié par la requête AT. A ce point d'arrêt peut être attaché une macro-requête et l'arrêt entraîne l'exécution des requêtes de mise au point appartenant à la macro-requête.

* Soit lorsque le simulateur a pris en compte le nombre d'instructions du système OBJET spécifié par la requête NUMBINST

- Le mode de travail est le mode semi-autonome.

Le couplage machine ESPION-machine OBJET était temporairement désactivé et le système testé est exécuté par l'unité centrale de la machine OBJET. L'arrêt de cette exécution et la réactivation du couplage sont provoqués par la rencontre d'un point d'arrêt ou par l'apparition d'une condition particulière.

* Un point d'arrêt est précisé par la requête BREAK.

* Une condition particulière est définie par la requête WHEN suivie de l'une des quatre directives : NORMAL, TRACE, SPY, STOP. Seule la directive STOP provoque l'activation du mode conversationnel sur occurrence de la condition particulière.

Une interruption de type externe (fonction console EXT) peut être un moyen d'arrêter une exécution erronée. Il est judicieux de définir les différents paramètres relatifs à ce type d'interruption de façon à ce que le mode conversationnel soit toujours réactivé.

IV-2-2 - Abandon du mode conversationnel

Le programmeur souhaite relancer l'exécution du système OBJET ; il aura auparavant indiqué quels sont les types de couplage qui doivent rester actifs durant cette exécution.

Si le couplage unité centrale est toujours présent, alors l'abandon du mode conversationnel se fait au profit du mode simulation qui est alors initialisé.

Si le couplage unité centrale est désactivé, alors c'est le mode semi-autonome qui est démarré.

La requête GO permet donc d'abandonner le mode conversationnel. Dans tous les cas, l'état du calculateur est restauré soit comme donnée du simulateur, soit dans l'unité centrale OBJET elle-même.

La requête LOGSPY permet elle aussi d'abandonner ce mode de travail. La machine ESPION est détruite, le couplage est définitivement désactivé. La machine OBJET redevient une machine virtuelle quelconque. Aucun contrôle sur l'exécution n'est possible.

IV-2-3 - Le langage de requêtes

Une requête est l'instruction de base du système de contrôle SPY . Elle se compose d'un mot clé qui est le nom de la requête et d'une suite d'arguments. Le blanc est le séparateur. Chaque mot clé admet une abréviation minimum ; une seule lettre suffit s'il n'y a pas de confusion possible.

Certaines requêtes ont pour arguments une liste d'options. Dans cette liste l'ordre n'est pas défini. En cas de contradiction, c'est le dernier argument donné qui est significatif. Les arguments admettent aussi des abréviations de quatre lettres au minimum.

Toutes les adresses qui figurent dans les requêtes font référence à la mémoire de la machine OBJET ; c'est-à-dire qu'il est inutile de leur affecter un facteur de translation pour avoir une adresse dans la mémoire de la machine OBJET.

Lorsque le couplage est actif, les fonctions consoles du système CP-67 font référence à des adresses dans l'ensemble de la machine ESPION . Dans ce cas il faut affecter un facteur de translation pour repérer les mots mémoires OBJET. Par contre si le couplage est temporairement désactivé les fonctions consoles font directement référence à la mémoire de la machine OBJET.

Nous donnons ci-dessous une liste de l'ensemble des requêtes de base disponibles dans le système de contrôle de SPY et un classement de ces requêtes suivant le mode de couplage nécessaire à leur fonctionnement. Il est évident que l'émission de ces requêtes rend obligatoire la présence de l'inclusion des unités centrales, puisque seule celle de la machine ESPION est à même de les traiter.

IV-2-3-1 - Requêtes permettant l'examen et la modification de la machine OBJET

Elles autorisent l'examen et la modification de la mémoire et de certains éléments priviligiés de l'unité centrale de la machine OBJET . Elles ne provoquent pas l'exécution du système testé dont l'état est figé.

X, CAW, CSW, DUMP, *X, * GPR	pour examiner la mémoire OBJET
PSW, GPR	pour examiner des éléments de l'unité centrale OBJET.
STORE, SET	pour modifier la mémoire OBJET et certains éléments de l'unité centrale OBJET
DUMP	pour obtenir des images partielles ou complètes de la mémoire OBJET.

IV-2-3-2 - Requêtes permettant le contrôle de l'exécution

Ces requêtes autorisent le contrôle de l'exécution du système résidant dans la machine OBJET . Elles sont regroupées en deux classes suivant quelles nécessitent l'utilisation du mode simulation ou l'utilisation du mode semi-autonome.

La première classe permet à l'unité centrale ESPION de contrôler le fonctionnement de la machine OBJET . Ces requêtes ne peuvent être prise en compte que si le simulateur d'instruction est actif.

STOP REFERENCE	Contrôle les adresses générées par les instructions du système testé. Chaque fois que l'une d'entre elles fait référence à une zone mémoire pré-définie le mode conversationnel est activé.
STOP INSTRUCTION	A le même but que le STOP REFERENCE mais le contrôle se fait sur le code opération des instructions à simuler.
AT	Définit des points d'arrêt permanents. On peut y associer des macro-requêtes qui ne seront exécutées que lors de la simulation de l'instruction se trouvant à l'adresse précisée.
ON	Définit les actions à entreprendre par le simulateur avant de réfléchir un type d'interruption asynchrone particulière que l'on précise dans la requête.
TRACE, COLLECT	Permettent de recueillir des renseignements sur l'exécution des instructions soit immédiatement (TRACE), soit à postériori (COLLECT).

La seconde classe autorise un contrôle de l'exécution du système testé, alors que le mode semi-autonome est actif.Le couplage de deux machines qui avait été provisoirement désactivé est rétabli sous certaines conditions précisées dans ces requêtes. Elles fixent pour différents types de conditions privilégiées, interruptions asynchrones par exemple, le mode de travail qui doit être mis en fonction.

BREAK	Permet de préciser les points d'arrêts. Lorsque l'instruction mentionnée dans la requête sera exécutée le couplage sera remis en fonction et le mode conversationnel activé.
WHEN	Fixe les conditions (telles que interruptions asynchrones ou instructions privilégiées) qui réactiveront le couplage. Des directives, données en paramètres précisent le mode de travail qui sera activé (mode conversationnel (STOP), mode simulation (SPY), mode semi-autonome (NORMAL)).

LISTE DES REQUETES DISPONIBLES DANS LE MODE CONVERSATIONNEL

I - LISTE DES REQUETES PROPRES AU SYSTEME SPY

IPLSPY	Permet de contrôler le chargement d'un système de programmation dans la mémoire OBJET.
MONITOR	Permet de préciser lequel des deux modes de travail -mode simulation ou mode semi-autonome- doit être activé.
WHEN	Précise les conditions qui permettent d'arrêter le mode semi-autonome.

II - LISTE DES REQUETES DE SPY ACTIVANT DES OUTILS DE MISE AU POINT

(voir leur description détaillée à la référence 26).

ADD	Effectue l'addition de deux nombres hexadécimaux.
AT(.)	Permet la définition d'une macro-requête.
BREAK	Permet la pose d'un point d'arrêt reconnu en mode semi-autonome.
BLIP	Compte les instructions.
CAW	Imprime sur le terminal le contenu de la mémoire d'adresse hexadécimale 48 de la mémoire Objet.
CLEAR	Annule certaines requêtes.
CLOSE	Ferme les unités de sortie.
COLLECT	Construit un historique de l'exécution.
CSW	Imprime sur le terminal le contenu de la mémoire d'adresse hexadécimale 40 de la mémoire Objet.
DECHEX	Assure les conversions de décimal en hexadécimal.
DUMP	Imprime une image de la mémoire Objet sur l'imprimante de machine Espion.
GO	Permet de quitter le mode conversationnel.
GPR	Imprime sur le terminal les registres de l'unité centrale Objet.

HEXDEC	Assure les conversions de l'hexadécimal en décimal.
LIST	Donne la liste de certaines requêtes demandées.
NUMBINST	Indique le nombre d'instructions à simuler.
ON	Définit l'action du simulateur sur apparition d'une interruption asynchrone.
ORIGIN	Indique une valeur absolue qui sera additionnée à toutes les adresses fournies dans les requêtes.
PRINT	Définit l'unité de sortie des informations.
PSW	Imprime sur le terminal le mot d'état programme de l'unité centrale Objet.
READ	Permet de lire une requête sur le terminal.
.SET	Permet la modification de certaines parties privilégiées de la machine Objet (registres généraux, mot d'état programme etc...).
STORE	Permet de modifier la mémoire de la machine Objet.
SUBTRACT	Effectue la soustraction de deux nombres hexadécimaux.
TRACE	Permet de tracer les instructions exécutées.
X	Permet d'examiner la mémoire de la machine Objet.
*GPR	Imprime sur le terminal le contenu de la mémoire Objet pointée par un registre de l'unité centrale Objet.
*X	Simulaire à *GRP mais la mémoire est adressée par un pointeur de la mémoire Objet.
STOP	Permet de préciser des conditions d'arrêt dans le mode simulation.

Requête_IPLSPY

Format :

IPLSPY XXX

XXX est l'adresse de l'unité de la machine OBJET à partir de laquelle doit se faire le chargement du système de programmation.

Utilisation :

Cette requête simule la fonction électronique de chargement initial (IPL). Elle construit dans la zone mémoire OBJET d'adresse 0 un mot d'état programme. Cette quantité est lue sur l'unité dont l'adresse est donnée en paramètre, et permet de commencer le chargement proprement dit. Celui-ci a lieu sous le contrôle du système SPY , le mode simulation ou le mode semi-autonome est activé.

Réponses :

1 - UNIT NOT FOUND
L'unité précisée en paramètre n'est pas une unité de la machine

2 - INVALID IPLSPY
Mauvaise initialisation de cette requête.

3 - IPL SIO ERROR
Le chargement initial ne peut se faire à partir de l'unité donnée.

4 - BAD IPL FUNCTION

Une erreur est apparue au cours du chargement initial, le mot d'état programme se trouvant à l'adresse 0 de la mémoire OBJET n'est pas significatif.

Exemples :

- IPLSPY 00C
  ipl sio error

- IPLSPY 190

- PSW
  01000190 00012008 — mot d'état programme construit par la requête .

- MONITOR ON — exécution du chargement a lieu en mode simulation.

- GO

Requête MONITOR

Format :

MONITOR	ON
	OFF

ON active le mode simulation. Le couplage est en fonction durant l'exécution du système testé, le simulateur d'instruction est actif.

OFF active le mode semi-autonome. Le couplage est provisoirement désactivé. Toute condition anormale est réfléchie à la machine ESPION.

Utilisation :

L'initialisation du système SPY ne permet de mettre en fonction que le mode conversationnel.
L'utilisateur doit décider quel sera le mode de fonctionnement lorsqu'il quittera le mode conversationnel par la requête GO.
La requête MONITOR peut être émise à tout instant permettant ainsi une bascule entre le mode simulation et le mode semi-autonome.
Sur abandon du mode conversationnel on activera soit le mode simulation (MONITOR ON) soit le mode semi-autonome (MONITOR OFF).

Requête WHEN

Format :

```
        | EXTERNAL
        | SVC
        | PROGRAM     <option1,<...>>
WHEN    | MACHECK
        | IO
        | PRIVOP
```

Le premier argument indique le type de la condition particulière et option 1,... indique les directives choisies. Les directives sont :

NORMAL — Le mode semi-autonome reste actif. Le programmeur ne désire pas être informé de l'occurrence du type de condition choisie. Cette directive ne peut être combinée avec d'autres.

TRACE
NOTRACE — Les circonstances de l'apparition de la condition sont imprimées (ou non) sur l'unité d'entrée-sortie activée par la requête PRINT.

SPY — L'apparition de la condition choisie rétablit le mode simulation. Tous les outils disponibles dans ce mode et qui ont été initialisés auparavant sont à nouveau pris en compte.

STOP — Le mode conversationnel est activé sur l'occurrence de la condition choisie.

Utilisation :

Cette requête n'a d'effet que lorsque le mode semi-autonome est en fonction. C'est le mode de fonctionnement le plus utilisé.
Les différentes options ont pour but de sélectionner les conditions de contrôle à retenir quand le système testé s'exécute librement.
Lorsque le mode conversationnel est abandonné, une séquence particulière réinitialise le système SPY en vue d'une reprise éventuelle de contrôle. Le système de la machine OBJET ne peut détruire cette réinitialisation.
Les options par défaut pour toutes les conditions sont STOP et TRACE.

Remarque :

Il ne faut pas confondre cette requête et la requête ON. Cette dernière n'est effective que lorsque le mode simulation est en fonction. Elle permet de donner à l'utilisateur une plus grande souplesse sur la simulation des interruptions.

```
S PY
_ ipl spy 190
_ psw
0 000019000012008
_ x 0 x18
A DDR    =    000000      00000190 00012008
_ monitor on
_ t race on tail alli
_ numbinst 1
_ go

   012008   05F0          BALR   R1=15  R2=0

         *15 4001200A

 _monitor off
_ when priv stop trace
_ go

   PRIVILEGED OPERATIONIN 01000002 800121F6

   0121F2   9C00E000      SIO                B1=14 D 1=000 (000190)

_when priv normal
_ when io stop trace
_go

   012 1FA   8200F280      LPSW               B1=15 D 1=280 (012288)

  ** *PSW CHANGES*** SYSTEM MASK=11111110     AMWP=0110

 INTERRUPT I/O      AT FE060190 00000000

_ psw

0 0000000000121FE
_
```

exécution en mode semi-autonor
définition des directives pour les instructions privilegiées.

simulation de ce type d'instruction

réfléchit une interruption entrée - sortie

nouveau mot d'état program après interruption

IV-3 - ABANDON DU SYSTEME SPY

L'utilisateur peut, quand il le désire, abandonner définitivement le système de contrôle SPY . La machine OBJET redevient une machine virtuelle quelconque.

La requête LOGSPY permet de quitter le mode conversationnel en détruisant la relation de couplage entre la machine OBJET et la machine ESPION.

L'utilisateur pourra, néanmoins, demander à nouveau l'aide du système SPY en émettant la fonction console SPY.

Requête LOGSPY

Format :

```
LOGSPY
```

Utilisation :

Cette requête permet d'abandonner définitivement le système de contrôle SPY . Le couplage entre les deux machines est détruit. La machine OBJET redevient une machine virtuelle quelconque.

Exemple :

```
SPY
_psw
0000019000012008
_monitor on
_trace on tail alli
_numbinst 1
_go


   012008   05F0          BALR   R1=15  R 2=0
        *15 4001200A

_logspy                                                    désactivation de la machine
                                                           ESPION
CMS..VERSION 3.0 janvier  72


 P (191): 50 FILES; 831 REC IN USE, 665 LEFT (of 1496), 56% FULL (10 CYL)
R; T=0.03/0.19 08.36.57

 VP
set adstop 5a8

 begin
ADDRESS STOP AT 000005A8                                   arrêt du système CMS
                                                           sur interruption entrée-sortie
VP
spy

 -SPY READY                                                Création de la machine
_psw                                                       ESPION
000400B3400005A8
_
```

IV - 3 - UN EXEMPLE D'UTILISATION

```
vp-67 V3.L1 a volde service

1 archives
ENTER PASSWORD:
********
READY AT 10.01.50 ON 05/27/00
VP
query virtual
 CORE - 00768K
 009 CONSL 050
 010 SPOOL RDR
 00C SPOOL RDR
 00D SPOOL PUN
 020 SPOOL PRT
 00E SPOOL PRT
 0FF RPQ TIMER
 190 2314- 290 054 CYL R/O VCPD1
 191 2314- 290 010 CYL R/W VCPD1
 290 2314- 290 203 CYL R/W VCPD1
 380 2314- 290 003 CYL R/O VCPD1

spy
 -SPY READY
_VP
query virtual
 SEGMENT  00 00512K
 SEGMENT  01 00768K
 009 CONSL 050
 010 SPOOL RDR
```

Création de la machine OBJET

Configuration de la machine OBJET

Activation du système SPY

```
00C SPOOL RDR
00D SPOOL PUN
020 SPOOL PRT
00E SPOOL PRT
0FF RPQ TIMER
0EC SPOOL RDR
0ED SPOOL PUN
0EE SPOOL PRT
190 2314- 290 054 CYL R/O VCPD1
191 2314- 290 010 CYL R/W VCPD1
290 2314- 290 203 CYL R/W VCPD1
380 2314- 290 003 CYL R/O VCPD1
```

Configuration de la machine ESPION

```
begin
SPY
_iplspy 192
 UNIT NOT FOUND
_iplspy 00c
IPL UNIT ERROR, CSW=00000000 02000000, SENSE=40
_iplspy 190
```

Chargement contrôlé du système CMS

```
_psw
0000019000012008
_monitor off
```

Choix du mode semi-autonome

```
_trace on tail alli
```

Options pour la trace des instructions privilégiées

```
_when priv stop trace
_when io stop trace
_go
  PRIVILEGED OPERATION IN 01000002 800121F6

  0121F2   9C00E000      SIO                B1=14 D1=000 (000190)
```

```
_go
  PRIVILEGED OPERATION IN 01000002 800121FE

  0121FA    8200F280       LPSW                    B1=15 D1=280 (012288)
 ***PSW CHANGES*** SYSTEM MASK=11111110            AMWP=0110
_go
 INTERRUPT I/O       AT FF060009 00000000
_trace off
_go
  PRIVILEGED OPERATION IN 01000002 A00121FE
_go
 INTERRUPT I/O       AT FF060190 00000000
_go
  PRIVILEGED OPERATION IN 01000002 A00121F6
_trace on
_when priv normal                    Sur instruction privilegiée, activation
_go                                  du mode semi-autonome

   0121FA    8200F280       LPSW                   B1=15 D1=280 (012288)
  ***PSW CHANGES*** SYSTEM MASK=11111110           AMWP=0110
  INTERRUPT I/O       AT FE060190 00000000
 _go

   0121F2    9C00E000       SIO                    B1=14 D1=000 (000190)
  ***PSW CHANGES***                                        CONDITION CODE=0

   0121FA    8200F280       LPSW                   B1=15 D1=280 (012288)
  ***PSW CHANGES*** SYSTEM MASK=11111110           AMWP=0110
  INTERRUPT I/O       AT FE060190 00000000
 _go

   0121F2    9C00E000       SIO                    B1=14 D1=000 (000190)
```

```
***PSW CHANGES***                                    CONDITION CODE=0

 0121FA   8200F280      LPSW                  B1=15 D1=280 (012288)
***PSW CHANGES*** SYSTEM MASK=11111110        AMWP=0110
 INTERRUPT I/O      AT FE060190 00000000
_trace off
_go
 INTERRUPT I/O      AT FE060190 00000000
_when priv spy
_trace on
_numb 2
_at 121fa
->x 78 x18
->read
go

  0121F2   9C00E000      SIO                   B1=14 D1=000 (000190)
 ***PSW CHANGES***                                    CONDITION CODE=0

  0121F6   4770F1EA      BC          X2=0      B2=15 D2=1EA (0121F2)   TEST CONDITION CODE

monitor off
when priv spy trace
go
  PRIVILEGED OPERATION IN 01000002 800121FE
AT 0121FA   01
X 78 XL8
ADDR    =    000078      00000000 000121FE
READ
go
```

Sur instruction privilegiée, activation du simulateur.

Point d'arrêt permanent et macro-requête associée

Réactivation du simulateur et prise en compte des outils dynamiques.

```
 0121FA   8200F280      LPSW                  B1=15 D1=280 (012288)

 ***INTERRUPT**** INPUT/OUTPUT    OLD PSW FE06019000000000 - NEW PSW 00000000000121FE
 ***PSW CHANGES*** SYSTEM MASK=00000000
STOP ON INTERRUPT I/O
when priv normal
trace off
break 1 121fa                                  point d'arrêt non permanent
break 2 12216                                  en mode semi-autonome
monitor off
go
SPY      ENTERED
BREAKPOINT 02 AT 012216
go
SPY      ENTERED
BREAKPOINT 01 AT 0121FA
go
 INTERRUPT I/O      AT FE060190 00000000
psw
00000000000121FE
monitor on
count on                                       Compteur d'instructions simulées
trace on tail bronly
stop inst 9c                                   Stop sur exécution d'instructions
stop inst 83                                   particulières.
numb 20
on i/o stop trace
on prog stop trace
list inst
```

Listes de certains outils dynamiques.

Examen d'un mot mémoire

OBJET 1) avec SPY

2) avec CP

```
STOP INSTRUCTION: 9C (SIO )
                  83 (DIAG)
stop refe 78 7c
x 78 x18
ADDR    =    000078      00000000 000121FE
VP
x 100078-10007c
?VPARG: 01

d 100078-10007c
100078 =  00000000   000121FE

begin
SPY
go
```

Trace uniquement des instructions de déroutement

```
  012202   4770F1F2      BC              X2=0    B2=15 D2=1F2 (0121FA)   TEST CONDITION CODE=

  01220A   4710F210      BC              X2=0    B2=15 D2=210 (012218)   TEST CONDITION CODE=

  012212   4770F1F2      BC              X2=0    B2=15 D2=1F2 (0121FA)   TEST CONDITION CODE=

T 012216   07FA          BCR             R2=10                           TEST CONDITION CODE=

T 01210A   4770F126      BC              X2=0    B2=15 D2=126 (01212E)   TEST CONDITION CODE=

T 012132   4680F158      BCT    R1=8     X2=0    B2=15 D2=158 (012160)
        *R8 00000002

T 012174   4780F182      BC              X2=0    B2=15 D2=182 (01218A)   TEST CONDITION CODE=
```

Trace rapide et abrégée

```
_trace on fast bronly
_go
  012190   4720F1A2      BC
T 012194   4630F160      BCT
T 012174   4780F182      BC
  012190   4720F1A2      BC
T 012194   4630F160      BCT
T 012174   4780F182      BC

_numb flow
_trace off
_go
STOP REFERENCE 000078 00007F AT 0121EA
_x 78 x18
ADDR    =    000078      00000000 000121FE
_go
STOP INSTRUCTION: 9C (SIO ) AT 0121F2
_go

 ***INTERRUPT**** INPUT/OUTPUT    OLD PSW FE06019000000000 - NEW PSW 00000000000121FE
STOP ON INTERRUPT I/O
_go
STOP REFERENCE 000078 00007F AT 01210E
_x 78 x18
ADDR    =    000078      00000000 000121FE
_go
STOP INSTRUCTION: 9C (SIO ) AT 0054D0
_go
STOP INSTRUCTION: 9C (SIO ) AT 0054E8
_go
STOP INSTRUCTION: 9C (SIO ) AT 0054E8
```

```
_go

STOP INSTRUCTION: 83 (DIAG) AT 003694
_numb 2
_trace on tail alli
_monitor on
_go

  003694   83480028      DIAG   I2=72          B1=0  D1=028 (000028)   BIT I2=01001000

  003698   12FF          LTR    R1=15  R2=15
 ***PSW CHANGES***                                    CONDITION CODE=0

_logspy
CMS..VERSION 3.0 janvier 72

P (191): 50 FILES; 831 REC IN USE, 665 LEFT (of 1496), 56% FULL (10 CYL)
R; T=0.02/0.17 11.49.55

VP
set adstop 5a8

begin
ADDRESS STOP AT 000005A8
VP
spy
 -SPY READY
_monitor off
_when priv normal
_trace on
_when io normal
```

Abandon du système SPY et poursuite de l'exécution.

Re-création du système SPY

Fonctionnement mode semi-autonome, avec trace des instructions privilegiées.

```
SPY

_go

  0028A2    82000000       LPSW                    B1=0  D1=000 (000000)

  00AB2E    9C0000FF       SIO                     B1=0  D1=0FF (0000FF)
 INTERRUPT SVC       AT 000400CA 400095E8
_break 1 28a2

_go

  0054D0    9C00F000       SIO                     B1=15 D1=000 (000191)

  0054DC    9D00F000       TIO                     B1=15 D1=000 (000191)
 ***PSW CHANGES***                                          CONDITION CODE=1

  0054E8    9C00F000       SIO                     B1=15 D1=000 (000191)

  0054DC    9D00F000       TIO                     B1=15 D1=000 (000191)
 ***PSW CHANGES***                                          CONDITION CODE=0

  0054E8    9C00F000       SIO                     B1=15 D1=000 (000191)

  0054F4    9D00F000       TIO                     B1=15 D1=000 (000191)
 ***PSW CHANGES***                                          CONDITION CODE=1

  0054F4    9D00F000       TIO                     B1=15 D1=000 (000191)
 ***PSW CHANGES***                                          CONDITION CODE=0

  003694    83480028       DIAG   I2=72            B1=0  D1=028 (000028)   BIT I2=01001000

  003694    83480028       DIAG   I2=72            B1=0  D1=028 (000028)   BIT I2=01001000
```

```
  003694    83480028       DIAG    I2=72           B1=0  D1=028 (000028)   BIT I2=01001000

  003694    83480028       DIAG    I2=72           B1=0  D1=028 (000028)   BIT I2=01001000

  003694    83480028       DIAG    I2=72           B1=0  D1=028 (000028)   BIT I2=01001000
SPY       ENTERED
BREAKPOINT 01 AT 0028A2
_gpr 0 15
00000015
0000984C
00000000
00000000
000095E2
0003F580
00000000
0003E8B8
0000954E
00012000
0003C378
00000001
40009540
800095B4
400098F4
00000000
_break 1 d086¢break 1 289e
_go

  0028A2    82000000       LPSW                    B1=0  D1=000 (000000)
 INTERRUPT SVC       AT 000400CA 6000960C
_go
```

```
INTERRUPT SVC      AT 010400CA 4000ABF0

_go

SPY      ENTERED

BREAKPOINT 01 AT 00289E

_go


 0028A2   82000000      LPSW                     B1=0  D1=000 (000000)

***PSW CHANGES*** SYSTEM MASK=00000001

INTERRUPT SVC      AT 010400CA 6000AC28

_go


 003694   83480028      DIAG    12=72            B1=0  D1=028 (000028)   BIT 12=01001000

 003694   83480028      DIAG    12=72            B1=0  D1=028 (000028)   BIT 12=01001000

 003694   83480028      DIAG    12=72            B1=0  D1=028 (000028)   BIT 12=01001000

 003694   83480028      DIAG    12=72            B1=0  D1=028 (000028)   BIT 12=01001000

 003694   83480028      DIAG    12=72            B1=0  D1=028 (000028)   BIT 12=01001000

 003694   83480028      DIAG    12=72            B1=0  D1=028 (000028)   BIT 12=01001000

 003694   83480028      DIAG    12=72            B1=0  D1=028 (000028)   BIT 12=01001000

 003694   83480028      DIAG    12=72            B1=0  D1=028 (000028)   BIT 12=01001000

 003694   83480028      DIAG    12=72            B1=0  D1=028 (000028)   BIT 12=01001000

 0028A2   82000000      LPSW                     B1=0  D1=000 (000000)
```

```
***PSW CHANGES*** SYSTEM MASK=00000001                    CONDITION CODE=2

 INTERRUPT SVC      AT 010400CA 6000AC34

_trace off

_go

 INTERRUPT SVC      AT 010400CA 6000DFD0

_go

 INTERRUPT SVC      AT 010400CA 6000E050

_go

 INTERRUPT SVC      AT 010400CA 4000E1D4

_go

 INTERRUPT SVC      AT 010400CA 4000E1D4

_go

 INTERRUPT SVC      AT 010400CA 7000E4CC

_go

 INTERRUPT SVC      AT 010400CA 6000E51C

_go

 INTERRUPT SVC      AT 010400CA 4000E5[illegible]

_
```

Uniquement la trace des instructions SVC

C O N C L U S I O N

Il est surprenant de constater dans la littérature informatique que de nombreux auteurs émettent des "voeux pieux" en ce qui concerne les principes de réalisation d'un environnement de mise au point. Mais en général, les aides à la mise au point existants ne mettent pas en application ces idées, faute de disposer d'un cadre approprié.

La construction d'un tel environnement peut se diviser en deux parties :

- d'une part concevoir des outils aussi efficaces que possible et d'un emploi commode.

- d'autre part implanter ces outils dans un cadre privilégié qui leur fournit des conditions optimales de fonctionnement et qui augmente leur puissance grâce à une disponibilité permanente.

Cette dualité a été mise en oeuvre, dans des applications actuelles, pour des environnements intégrés à des systèmes de programmation et destinés à la mise au point de "programmes utilisateurs". Par contre, il semble qu'elle ait été oubliée lors de la construction de supports destinés à la mise au point des systèmes de programmation eux-mêmes.

Notre ambition de construire un support de mise au point destiné aux utilisateurs confrontés avec ce dernier problème -la mise au point des systèmes de programmation- nous a amené à concevoir et réaliser un système de contrôle capable de diriger l'exécution d'un ensemble de programmation quelconque.

De par la nature des différents travaux précédents concernant ce sujet, nous nous sommes surtout intéressé à la définition d'un environnement de travail et à la mise en place de relations entre le système de contrôle et le système de programmation en cours de test.

Le but de notre travail est de permettre à un programmeur système d'utilise les facilités fournies par un système en temps partagé et par un système conversationnel pour construire et réaliser un nouveau système de programmation.

Les principes de CP-67 éliminent un certain nombre de contraintes liées à ce type de réalisation, en particulier tout ce qui se rapporte aux essais sur un calculateur, puisque le programmeur système dispose d'une machine virtuelle où il peut implanter ses programmes.

En conséquence nous nous sommes fixés plusieurs objectifs :

- Réaliser et intégrer dans CP-67 un ensemble de programmes interactifs dont la finalité est de donner les moyens nécessaires à une mise au point rapide et précise. Nous utilisons pour cela un mécanisme qui offre un choix dans l'exécution contrôlée soit en utilisant un simulateur d'instruction, soit en laissant le programme s'exécuter normalement.

- Faire de l'ensemble SPY un outil facile à utiliser en insistant sur la simplicité des commandes et des fonctions.

- Permettre au programmeur d'utiliser SPY quelque soit ses programmes et sans que celà ait été prévu au cours de leur réalisation.

- Réduire le coût de la phase de mise au point d'un système de programmation.

Nous pensons avoir atteint ces objectifs, peut être pas toujours de façon parfaite, mais nous avons créé un produit opérationnel, déjà utilisé par certains programmeurs systèmes. Le fait d'avoir pu nous intégrer à l'équipe de maintenance du système CP-67 nous a permis de mieux définir les principes de base et les avantages de notre système et de mieux comprendre les problèmes qui se posent à des utilisateurs potentiels de SPY. De ce fait nous avons pu faire les premières expérimentations du système de contrôle pour déterminer des conditions anormales de fonctionnement des systèmes CMS et CMS-BATCH. La mise au point de ce dernier système étant particulièrement fructueuse puisqu'il ne disposait d'aucun support permettant de le tester.

Si le système de contrôle SPY est utile en lui-même nous pensons que les notions telles que couplage de deux machines virtuelles présentent un intérêt dans un cadre autre que celui de la mise au point. Elles permettent d'envisager de nouveaux moyens de communications entre machines virtuelles. Ce problème de communication est soulevé dans presque toutes les nouvelles applications qui sont développées à partir du concept de machines virtuelles. Nous croyons avoir apporté une première solution souple et puissante à cette question.

Nous pensons que l'ensemble de nos travaux donne aux utilisateurs des outils leur permettant de mieux comprendre les résultats de leurs expériences, d'utiliser avec une meilleure efficacité toutes les ressources de l'ordinateur sur lequel ils travaillent, d'aborder avec réalisme le délicat et rébarbatif travail de mise au point et de diminuer le coût d'une telle opération. Il doit aussi leur donner les moyens de mieux connaître le comportement réel de leurs programmes et donc de les améliorer. Ces quelques modifications apportées à CP-67 augmentent de façon sensible l'intérêt d'un tel système.

Avoir le désir d'apporter une solution à un des problèmes d'utilisateurs d'un calculateur est à la fois un but ambitieux et ingrat. Ambitieux car il suppose que l'on ait su comprendre les difficultés de ces programmeurs, ingrat car il augmente automatiquement les exigences de ceux-ci. Néanmoins, ces nouveaux besoins permettent de fixer de nouvelles directions de recherche et remettent toujours en cause ce qui existe déjà.

BIBLIOGRAPHIE

(1) F. AKIYAMA
An example of software system debugging.
Congrès de l'I.F.I.P. Août 1971 Volume Computer Software.

(2) F.C. BLAIR
An extensible interactive debugging system.
PURDUE University Computing Center - juin 1971.

(3) B. BUSSEL, R.A. KOSTER
Instrumenting computer system and their programs.
Fall Joint Computer Conference 1970.

(4) P.J. DENNING
Virtual memory.
Computing surveys Vol. 2 n° 3 September 1970.

(5) P.J. DENNING
Third generation computer systems.
Computing surveys Vol. 3 n° 4 Décembre 1971.

(6) J.B. DENNIS, E.C. VAN HORN
Programming semantics for multiprogrammed computations.
Communication of the A.C.M. Vol. 9 n° 3, mars 1966.

(7) W.R. DENISTON
SIPE A TSS/360 Software measurement technique.
Northumbrian Universities Multiple Access Computer Assembler Language Symbolic Debugging System.

(8) P. DEUTSCH, C.A. GRANT
A felxible measurement tool for software systems.
C-ngrès de l'I.F.I.P. Août 1971 - Volume Computer Software.

(9) E.W. DIJKSTRA
The structure of the "THE"-Multiprogramming system.
Communication of the ACM. Vol. 11 n° 5 , mai 1968.

(10) H.W. FLANAGAN
Program Monitoring Technique.
IBM Technical DISclosure Bulletin vol. 13, n° 8, 1971.

(11) K. FUCHI, H. TANAKA, Y. MANAGO, T. YUBA
A program simulator by partial interpretation.
Second symposium on operating systems principles 1969.

(12) C. HANS, A. AUROUX
Notion de machines virtuelles.
Revue de l'A.F.C.E.T. Décembre 1968, n° 15

(13) C. HANS, P. LEFEBVRE, B. PETEUL-HARMEL
Mise au point conversationnelle de systèmes grâce à un couplage particulier de machines virtuelles.
Canadian computer conference Session 72. juin 1972.

(14) IBM - Control Program-67/ Cambridge Monitor System Version 3
USER'S GUIDE. Manuel n° GH20-0859.

(15) IBM Control Program-67.
Program Logic Manual - manuel n° GY20-0590

(16) IBM System/360 Model 67.
Functional characteristics - manuel n° A27-2719.

(17) IBM System/360
Principles of operation - manuel n° A22-6821.

(18) M. JACOLIN
Accompagnateur FORTRAN : FORTBUG.
Note interne de l'IMAG - GRENOBLE mai 1970.

(19) N.J. KING
System Program Debugging.
IBM Technical Disclosure Bulletin vol. 13 n° 7 - 1970.

(20) B.W. LAMPSON
Dynamic protection structure.
Fall Joint Computer Conference. 1969.

(21) P. LEFEBVRE
Adressage symbolique dans PILOTE.
Note interne de l'IMAG - GRENOBLE - juin 1971.

(22) P. LEFEBVRE, B. PETEUL-HARMEL
PILOTE et SPY Manuel d'utilisation et description logique
Note interne de l'IMAG - mai 1972.

(23) J.P. LEHEIGET
Généralisation de la notion d'espace virtuel dans le système CP-67.
Thèse C.N.A.M. à paraître juillet 1972.

(24) R.A. MEYER, L.H. SEAWRIGHT
A virtual machine time sharing system.
IBM System Journal n° 3 - 1970.

(25) R. PERRONNEAU, C. GATEAU
Réalisation d'un simulateur de machine IBM 360 type 67.
Rapport DEA à paraître juin 1972 - IMAG.

(26) B. PETEUL-HARMEL
La mise au point de programmes par simulation. Réalisation d'un support conversationnel de mise au point : PILOTE.
Université Scientifique et Médicale de Grenoble - Thèse - juin 1971.

(27) R. SEDGEWICK, R. STONE, J.W. Mc DONALD
SPY A program to monitor OS/360.
Fall Joint Computer Conference 1970.

(28) E.C. VAN HORN
Three criteria for designing computing systems to facilitate debugging.
Communications of the A.C.M. Vol. 11 n° 5 - mai 1968.