THESE

présentée

A L'UNIVERSITÉ DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir

LE DIPLOME DE DOCTEUR DE 3e CYCLE

par

André DUCROT

Spécialité : INFORMATIQUE

PRINCIPES D'ORGANISATION ET DE RÉALISATION DU SYSTEME GRAPHIQUE INTERACTIF GIPSY

Soutenue le 22 mai 1974 devant la Commission d'examen composée de :

MM. P. BACCHUS		Président
C. CARREZ		Examinateur
V. CORDONNIER		Rapporteur
N. WISEMAN	}	Invités
A. LEMAIRE	}	

Je tiens à remercier

Monsieur le Professeur P. Bacchus qui m'a fait l'honneur d'accepter de présider le jury,

Messieurs les Professeurs C. Carrez et N. Wiseman qui ont accepté de participer au jury,

Monsieur le Professeur V. Cordonnier qui a bien voulu se charger du rôle de directeur de thèse,

Je remercie enfin Messieurs Lemaire, Prusker, Jancène, Saltel, Boullier et Gros, mes collègues du groupe graphique pour les nombreuses discussions, les encouragements et les critiques dont cet ouvrage a bénéficié.

I. GENERALITES SUR LES SYSTEMES GRAPHIQUES.

Nous appellerons système graphique l'ensemble des outils informatiques permettant l'utilisation d'un ou plusieurs terminaux de visualisation.

Nous pouvons distinguer deux types de terminaux graphiques :

- les terminaux où il est possible de modifier une partie d'image sans devoir régénérer l'image complètement. Ce sont par exemple, les consoles à rafraîchissement où l'image est stockée dans une mémoire qu'il est possible de lire et d'écrire.

- les terminaux où on ne peut pas modifier l'image affichée sur l'écran ; par exemple les tubes à mémoire où l'image est conservée sur le tube lui-même. L'avantage de ce type de terminaux est de permettre l'affichage d'images beaucoup plus complexes car les limitations des consoles à rafraîchissement (taille de la mémoire et temps de régénération) n'existent pas.

Un terminal graphique n'est pas un simple dispositif d'affichage. Il met à la disposition des utilisateurs un certain nombre de moyens de dialogue avec l'ordinateur.

Citons par exemple :

- pointeur optique,
- clavier de fonctions,
- clavier alphanumérique,
- tablette graphique
- manche à balai, boule de pointage ... .

Un système graphique doit donc permettre l'utilisation de ces moyens de dialogue.

Prenons un exemple d'application graphique :

Dans la conception de circuits électroniques, l'ordinateur peut effectuer l'analyse rapidement et avec beaucoup de précision, mais seul l'utili-

sateur au vu de la représentation schématique du circuit et des résultats des calculs peut prendre la décision de modifier la topologie du circuit.

Nous voyons donc dans cet exemple que c'est l'homme qui prend les décisions, c'est ce que nous appellerons une application interactive.

Distinguons les termes "interactif" et "conversationnel".

Nous dirons qu'un programme est conversationnel lorsque l'utilisateur peut interrompre à tout moment l'exécution pour demander des renseignements (valeurs de variables, "adresse" de l'instruction en cours...) ou pour modifier les valeurs de certaines variables ou l'"adresse" de la prochaine instruction à exécuter, le programme pouvant très bien se dérouler sans interruption de l'homme. Ce mode de travail est donc surtout utilisé au cours de la mise au point d'un programme.

Par contre, dans une application interactive c'est l'homme qui dirige l'exécution et l'ordinateur ne peut pas en certains points particulier continuer sans intervention de l'utilisateur. Ce mode de travail est généralement le mode de fonctionnement d'une application graphique.

Remarque: ces deux modes ne s'excluent absolument pas, ils sont même complémentaires.

Essayons de dégager quels sont les outils spécifiques que doit fournir un système graphique.

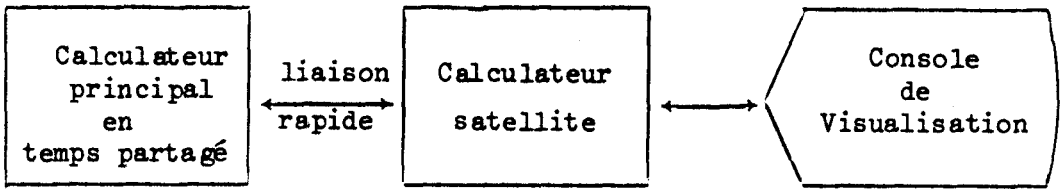
Il faut bien entendu générer des images.
Il faut permettre l'emploi des moyens de dialogue.
Enfin, il faut fournir des outils de structuration des données et des liaisons entre images et structures de données.

Reprenons l'exemple d'un circuit électronique :

Pour pouvoir décrire son circuit c'est-à-dire les éléments qui le composent et les liaisons entre ces éléments, le concepteur a besoin de construire une structure de données.

S'il désigne un élément à supprimer, le système doit lui permettre de retirer l'élément de l'image du circuit et surtout d'identifier cet élément afin de pouvoir effectuer les modifications dans la structure de données.

Une contrainte importante des systèmes graphiques est d'assurer un bon temps de réponse aux interventions de l'utilisateur. Pour satisfaire cette exigence, nous estimons que la configuration idéale pour un système graphique est la suivante :



Cette configuration présente de nombreux avantages. Elle permet de décharger le calculateur principal de la gestion de la console et de ses moyens de dialogue et de traiter l'ensemble calculateur satellite, utilisateur comme un usager normal du système. De plus, si le calculateur satellite est suffisamment puissant, il est possible de traiter certaines applications dans le calculateur satellite.

Il faut noter qu'une telle organisation est mieux adaptée pour traiter des problèmes de conception assistée que des problèmes de simulation en temps réel ou d'animation d'images complexes pour lesquels il faut souvent construire un système spécialisé où la gestion des moyens de dialogue est peu importante et la rapidité de modification des images est essentielle.

Nous nous intéresserons essentiellement dans ce travail au rôle du calculateur satellite, c'est-à-dire la gestion des images et des moyens de dialogue, mais avant, nous allons aborder brièvement les problèmes de génération d'images et de structuration des données car la façon dont ils sont résolus influence fortement la conception des outils nécessaires.

II. GENERATION D'IMAGES.

Il est évident que la génération d'images est le premier problème qui se pose lors de la réalisation d'un système graphique.

Il a été développé essentiellement deux techniques pour résoudre de problème :

- utilisation de listes d'affichage structurées,
- génération dynamique d'images.

La liste d'affichage est la suite des ordres envoyés au processeur graphique pour afficher et rafraîchir l'image. Certains systèmes utilisent une liste d'affichage structurée (souvent sous forme d'arbre).

Soit par exemple l'image suivante à représenter :

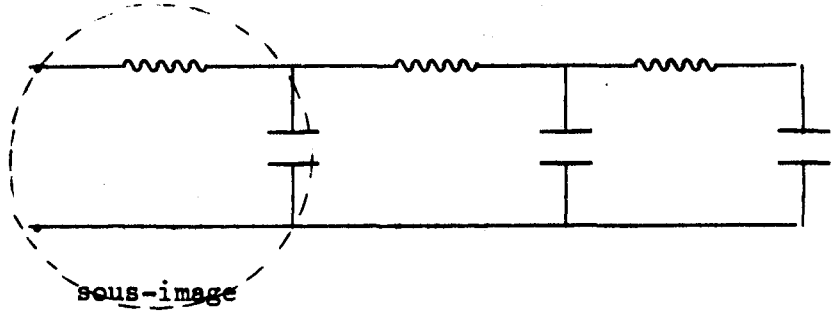


Fig. II.1

Il est possible alors d'adopter la structure suivante :

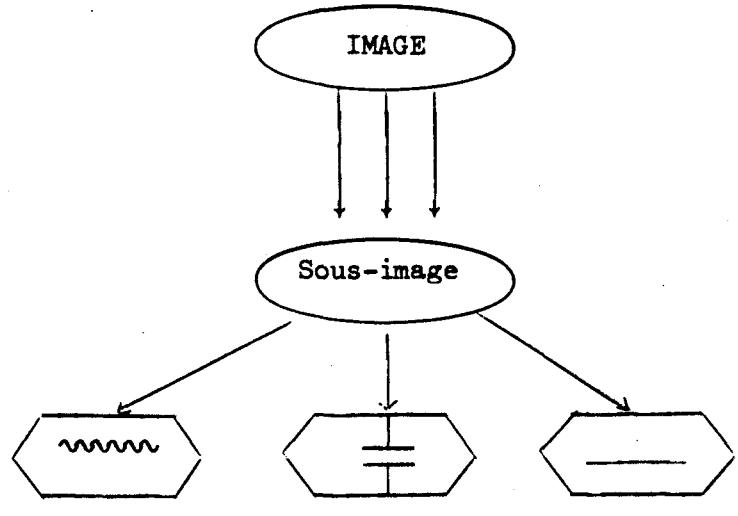


Fig. II.2

Les avantages de cette technique paraissent importants :

- il est relativement facile d'ajouter ou de retirer un élément à l'image.
- la récupération lors d'une désignation de la hiérarchie de l'image est aisée.
- la place occupée par l'image en mémoire de rafraîchissement est réduite grâce à l'utilisation possible des sous-programmes.

Elle présente cependant de graves inconvénients. Il y a souvent duplication (et même parfois "triplication") de l'information concernant l'image.

Soit par exemple à effectuer sur une image une transformation simple telle qu'une mise à l'échelle. L'information sera alors présentée sous trois formes dans le système.

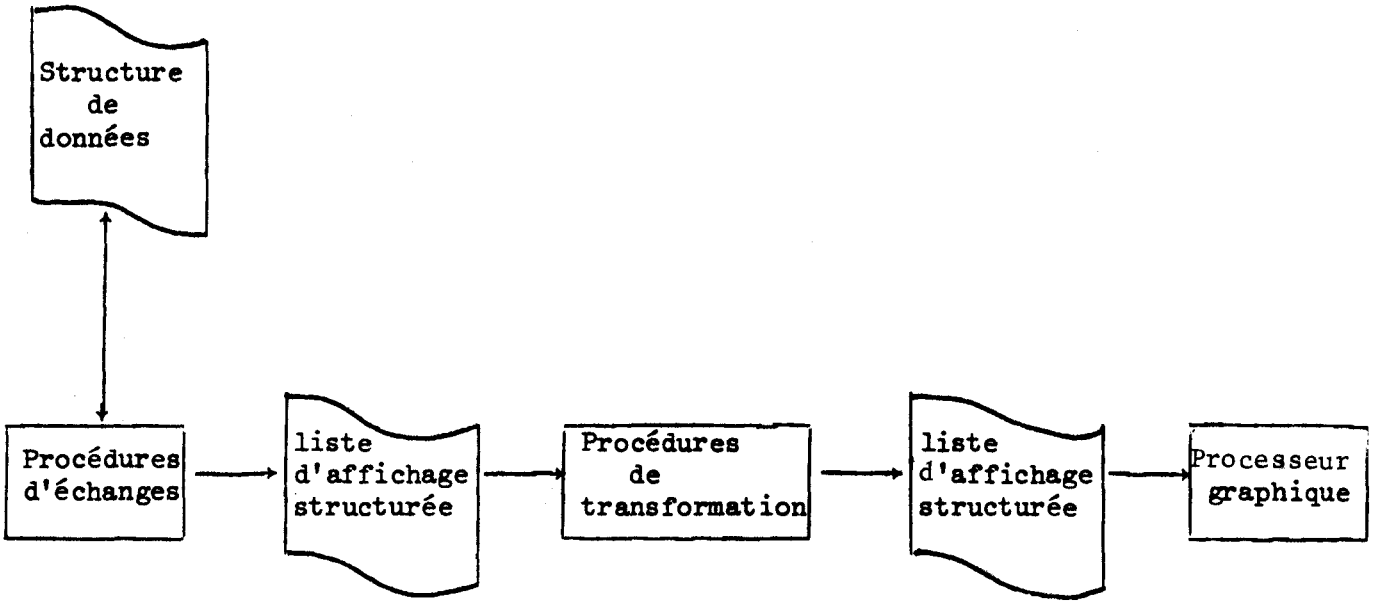


Fig. II.3

De la structure de données est extraite une liste d'affichage structurée et c'est cette structure qui est manipulée pour effectuer les transformations.

De plus, les systèmes utilisant cette technique, imposent souvent à l'utilisateur l'adoption d'une structure de données analogue à la structure

d'image, structure qui est généralement peu adaptée au type de données que l'utilisateur doit manipuler.

W.M. Newmann dans [1] a proposé une autre technique de génération d'images. L'image est générée dynamiquement et n'est pas structurée. Elle résulte de l'appel de procédures "graphiques". C'est la technique que nous avons utilisée (voir en particulier [2] pour l'extension graphique de CPL/1 du système METAVISU ou [3] pour le FORTRAN graphique 3 Dimensions). En fait nou n'utilisons pas la notion de procédure graphique mais plutôt l'appel graphique de procédure. Un appel graphique de procédure est un appel de sous-programme comportant des options graphiques. Ces options permettent de préciser le positionnement de l'image, les transformations à lui apporter (rotation translation, mise à l'échelle, perspective, élimination de lignes cachées ...), la partie d'image à générer effectivement (découpage) ou d'associer un "nom" à l'image générée par la procédure.

Exemple :

```
call Routine (A); /* n'est pas un appel graphique de procédure */
call Routine (A) at 100,100; /* est un appel graphique de procédure
car il y a une option précisant le positionnement de l'image générée par
Routine (A) */
```

Ce mode de génération permet de donner à l'utilisateur pour décrire ses images, toutes les facilités des langages de programmation qu'il a l'habitude d'utiliser (possibilité de génération conditionnelle d'images ou de parties d'images, récursivité dans les appels graphiques de procédure, ...).

De plus, l'image générée lors d'un appel graphique de procédure est décrite dans un espace local à la procédure, ce qui permet à l'utilisateur de travailler dans le système de coordonnées qui lui convient. Il n'aura à préciser au plus bas niveau d'appel que les dimensions du masque (dans son espace propre) et de la fenêtre (sur l'écran) pour obtenir une image affichable. Le masque est la portion d'espace à visualiser et la fenêtre la dimension de la projection dans l'espace de visualisation.

Pour optimiser le temps de calcul, un appel graphique de procédure qui générerait une partie d'image entièrement "découpée" n'est pas effectué (boxing-process)

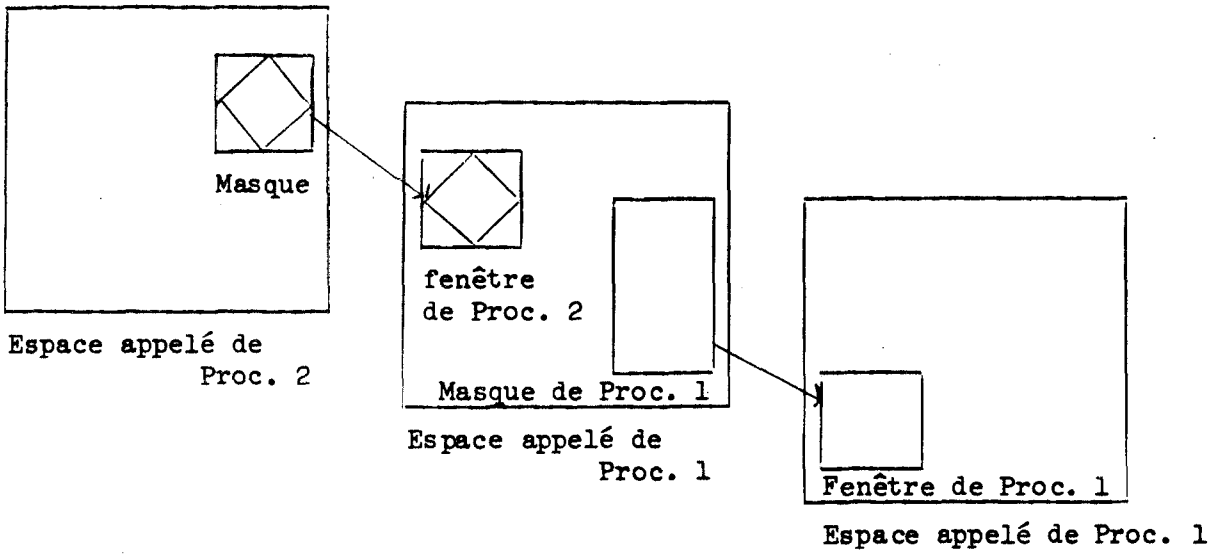


Fig. II.4

Enfin, ce mode de génération est indépendant du type de terminal utilisé.

Le principal inconvénient est que toute modification apportée à l'image oblige à la régénérer complètement, ce qui est généralement rapide. Remarquons que contrairement à W. Newmann, il n'est pas nécessaire dans notre système de régénérer l'image pour retrouver l'élément désigné (rôle des associateurs).

Signalons que la technique d'extension du FORTRAN que nous avons utilisée (par pré-processeur transformant les pseudo-instructions graphiques en appels de sous-programmes de bibliothèque) peut être aisément utilisée pour réaliser une extension graphique d'autres langages, ceci sans modification des compilateurs.

Exemple : Reprenons l'image que nous avons utilisée ci-dessus et décrivons-la en CPL/1 étendu

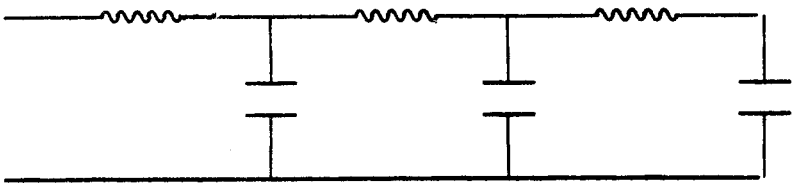


Fig. II.5

```
CIRCUIT : proc(N);
déclare N bin fixed (15); /* nombre de sous circuits */
déclare I bin fixed (15);
RESISTANCE : proc ;
line of 10,0 of 5,10; /* ligne brisée définie par des déplacements relatifs */

do I = 1, 4;
line of 10,-20 of 10,20;
end;
line of 5,-10 of 10,0
end RESISTANCE;
FIL : proc;
line of 10,0;
end FIL;
CAPACITE : proc;
line of 0,50: move of -10,0; /* déplacement relatif */
line of 20,0; move of -20,-10;
line of -20,0; move of -10,0;
line of 0,-50;
End CAPACITE;
SOUS-CIRCUIT : proc;
call RESISTANCE at 0,110;
call CAPACITE at 110,110;
call FIL at 0,0 scale 11,1; /* mise à l'échelle */
end SOUS-CIRCUIT;
DO I = 0 to N -1;
call sous-circuit At /10 * I,0;
end;
end CIRCUIT;
```

Pour effectuer la mise à l'échelle il suffira d'écrire

CALL CIRCUIT (B) At XO, YO scale X,Y;

III. STRUCTURATION DES DONNEES.

Les données sont formées de valeurs numériques, de chaînes de caractères..., liées entre elles par des relations.

Par exemple, dans un circuit électrique, il n'y a pas que les valeurs des différents composants qui soient importantes, mais aussi les connections entre ces différents composants.

Les données sont stockées dans la mémoire de l'ordinateur de façon organisée pour préserver les relations qui existent entre elles et aussi pour fournir un mode d'accès aux différentes données. L'organisation des données est appelée structure de données.

Le but d'un système interactif (pas nécessairement graphique) est de permettre des échanges efficaces entre l'homme et la machine, ce qui signifie que l'homme peut dialoguer avec la machine pendant l'exécution de son programme pour influencer le déroulement des calculs.

En fait, dans un système interactif, il s'agit de créer et de modifier dynamiquement un modèle du problème à résoudre. Ce schéma est résumé par l'équation suivante attribuée à D.T. ROSS :

MODELE = Données + Structure + Algorithme.

Dans un système graphique interactif, il est indispensable que des liaisons existent entre l'image, représentation schématique d'une partie de la structure de données et la structure.

Reprenons l'exemple d'un circuit électronique. Si l'utilisateur désigne un composant à supprimer, il faut faire la modification non seulement sur l'écran, mais aussi et surtout dans la structure. Il est donc nécessaire que le système fournisse un moyen de liaison entre l'élément désigné et son correspondant dans la structure de données.

III.1. DIFFERENTS TYPES DE STRUCTURES.

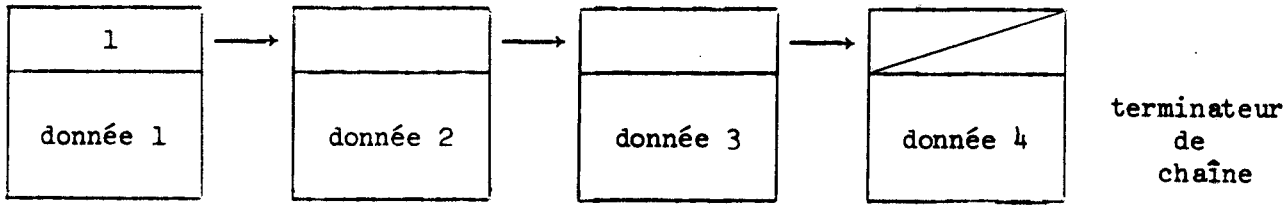
On peut considérer qu'il existe trois modes d'organisation possibles pour construire une structure de données : organisation séquentielle, répartition aléatoire et organisation sous forme de listes.

L'organisation séquentielle indexée utilisée pour les tableaux par exemple, est la plus simple à réaliser car le mécanisme d'accès aux données est toujours cablé (indexation). Dans cette organisation, les données sont stockées séquentiellement et ordonnées. Les données peuvent être retrouvées très rapidement mais l'insertion d'une nouvelle donnée ou la suppression d'une donnée est un processus lent et difficile car toute la suite des données doit être remise à jour. Pour cette raison, l'organisation séquentielle est souvent insuffisante pour les systèmes graphiques.

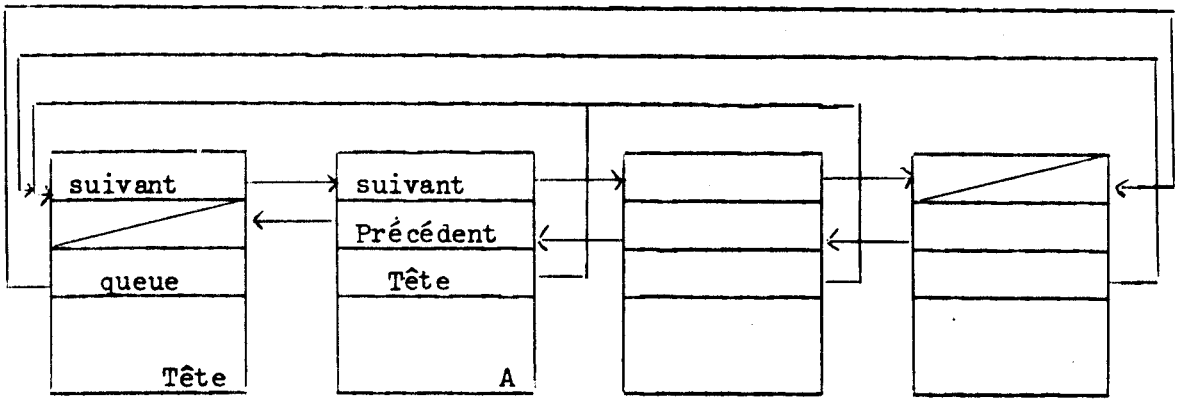
L'organisation sous forme de listes est une organisation dans laquelle les données sont chaînées entre elles par des pointeurs. Différentes solutions sont possibles suivant qu'on optimise l'accès ou la place en mémoire occupée par la structure.

Les structures plus complexes peuvent être construites à partir de ces structures de base. Les plus fréquemment utilisées sont les structures d'arbre : GRAPHIC 2, PL/1, CØBØL et les structures hiérarchisées (ASP, RSP).

Les modifications de la structure (insertion, suppression d'élément) sont rapides car elles ne nécessitent pas de mise à jour de l'ensemble de la structure mais l'accès à un élément peut être long.



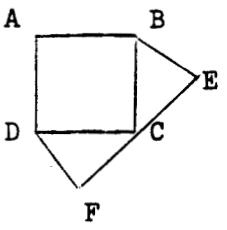
(a) liste à simple chaînage



(b) Structure d'anneau à double chaînage
et pointeur vers la tête

Fig. III.1

Exemple de structure hiérarchisée.



la structure utilisée peut être la suivante :

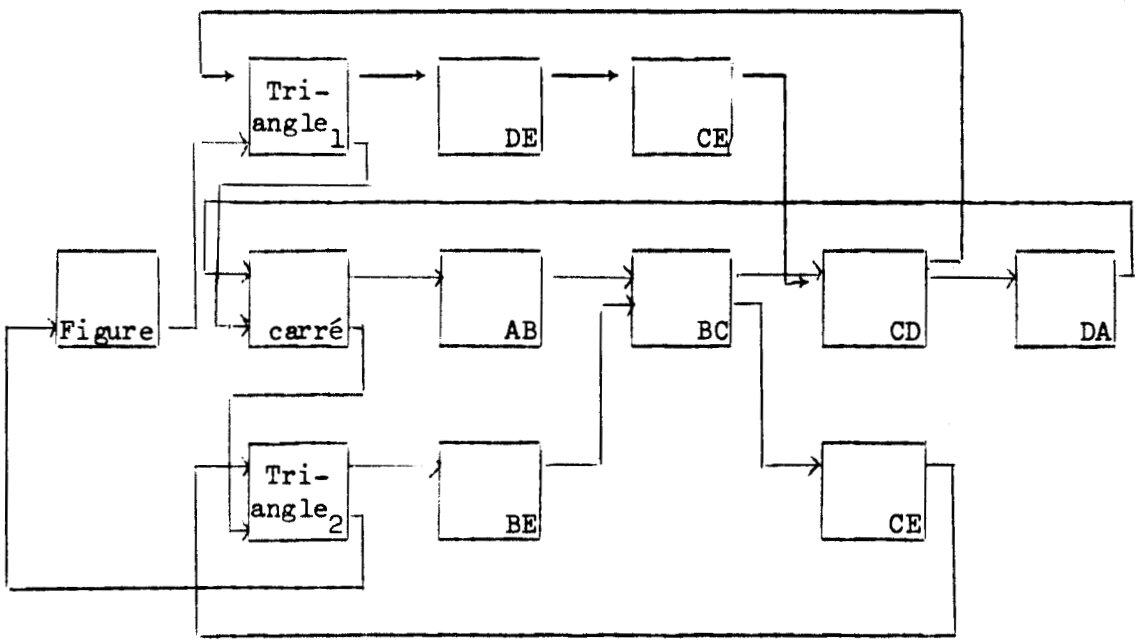


Fig. III.2

Par souci de clarté du schéma, les pointeurs vers la tête et vers l'élément précédent n'ont pas été représentés.

Ce type de structure a été très souvent utilisé dans les systèmes graphiques (Sketchpad, RSP, UNIVAC, par exemple).

Le troisième type d'organisation possible est la répartition aléatoire des données en mémoire. Une donnée est stockée à une adresse arbitraire et

on y accède grâce à son nom. La méthode la plus simple est d'avoir une table contenant le nom et l'adresse de la donnée. La table est appelée dictionnaire ou table de symboles. Pour des structures de données importantes la dimension de la table et donc le temps de recherche deviennent très longs.

Donc ordonner la table séquentiellement n'est pas une solution souhaitable car les modifications sont longues à effectuer. Une solution fréquemment utilisée est d'utiliser le nom, de le traiter comme un nombre et d'effectuer un calcul standard, puis d'utiliser le résultat du calcul comme adresse de la donnée (hash-code).

Exemple :

	Nom	L I N E 3
	Représentation Machine	21 18 23 14 03
Calcul du Hash-code	Somme	79
	Calcul de l'adresse (3ième caractère et somme)	23 79

Il peut se produire que plusieurs noms donnent la même adresse après calcul du Hash-code. Dans ce cas, on dit qu'il y a collision. Beaucoup de solutions ont été proposées pour résoudre ce problème. La plus couramment employée consiste à chaîner dans une liste tous les noms et les données correspondantes qui ont le même hash-code. Quand une référence à l'un de ces éléments est faite, il est nécessaire de calculer le hash-code, puis rechercher dans la liste, la donnée correspondante.

Un exemple de ce type d'organisation est la structure associative du langage LEAP [4] . Elle utilise des associations d'items appelés triplets que l'on peut interpréter comme

l'Attribut de l'OBJET est la VALEUR

exemple : le PERE de PAUL est JEAN

Les questions élémentaires que l'on peut poser à la structure sont données dans le tableau ci-après, la ou les inconnues étant notées par "?".

FORME	EXEMPLE	REPONSE
A O ?	PERE PAUL ?	les pères de Paul (1)
A ? V	PERE ? JEAN	les fils de Jean
? O V	? PAUL JEAN	les relations entre PAUL et Jean
A ? ?	PERE ? ?	les couples fils père
? O ?	? PAUL ?	les couples A-V des triplets où Paul est en 2ième position
? 2 V	? ? JEAN	les couples A-O des triplets où Jean est en 3ième position
? ? ?	? ? ?	tous les triplets de la structure

Fig. III.3

Pour le système METAVISU, nous avions réalisé une extension de CPL/1 pour la manipulation d'une structure associative de ce type [5]. L'utilisation de cette structure est aisée et la recherche et les modifications rapides (ceci grâce au mécanisme de pagination d'ESOPE).

III.2. PROBLEMES D'IMPLEMENTATION ET D'UTILISATION D'UNE STRUCTURE DE DONNEES DANS UN SYSTEME GRAPHIQUE .

Ces problèmes découlent de la répartition des tâches entre les deux calculateurs. S'il est évident qu'en général, la structure de données est implantée dans le calculateur principal, il semblerait intéressant de pouvoir manipuler dans le calculateur satellite une partie de la structure. En fait, ceci est difficilement réalisable car il est alors nécessaire de reporter dans la structure principale les modifications apportées au niveau local, ce qui peut être très complexe et très lourd (modifications d'éléments déjà modifiés, par exemple). De plus la struc-

ture locale est souvent insuffisante ce qui fait que le nombre d'appels au calculateur principal n'est pas nécessairement diminué.

Exemple : On sélectionnera dans la structure de données, la partie correspondant uniquement à l'image affichée sur l'écran. Si l'utilisateur désire accéder à des éléments non affichés (en raison du découpage par exemple) ou à des données numériques, il devra faire appel à la structure principale.

Enfin, une telle méthode ne pourrait s'appliquer qu'à une seule structure de données (la structure de données unique fournie par le système), structure qui pourrait ne pas être adaptée au problème de l'utilisateur.

Nous avons donc retenu les solutions suivantes :

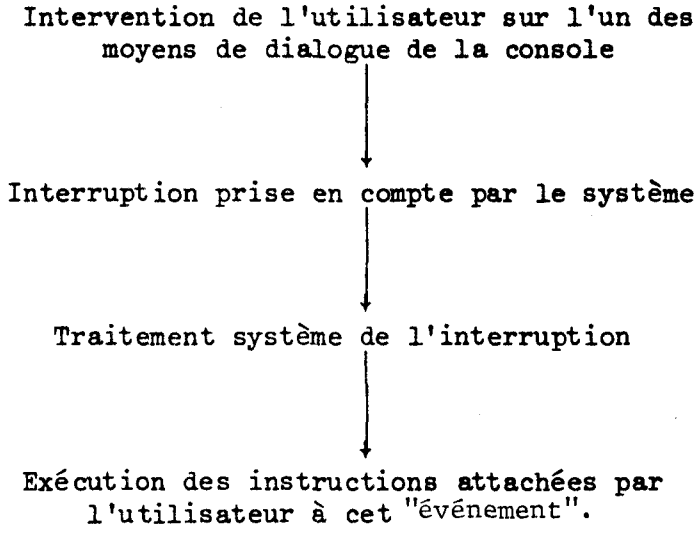
1/ Libre accès de l'utilisateur aux structures de données existantes dans le système du calculateur principal, l'utilisateur choisissant le mode de structuration qui lui convient.

2/ Pour permettre les échanges entre images et structure de données, possibilité d'inclure dans l'image des "associateurs" qui peuvent être reliés aux éléments de la structure et sont communiqués à l'utilisateur lors d'une désignation.

3/ Permettre de modifier immédiatement et localement l'image actuellement visualisée.

IV. PRINCIPES DE LA GESTION DES INTERACTIONS.

Il est possible de décrire une interaction comme étant la suite des actions suivantes :



Nous appellerons "événement" une action sur les moyens de dialogue et nous dirons que l'évènement est reconnu lorsque le système constate que cet évènement était prévu par le programme utilisateur.

Notons cependant qu'il existe des évènements sur lesquels l'utilisateur n'a pas d'action directe, par exemple :

- sablier (lié à l'horloge interne du calculateur ou au cycle de rafraîchissement de la console de visualisation).

Il y a plusieurs techniques couramment utilisées pour spécifier les actions à effectuer lors de sa reconnaissance d'évènement. De façon générale, le système collecte un certain nombre d'informations concernant l'état de la console et du programme lorsque l'interruption se produit. Ces informations sont mises à la disposition des procédures de traitement écrites par l'utilisateur. Les différences majeures entre les divers systèmes concernent les méthodes de spécification des procédures à exécuter lors de la

reconnaissance d'un évènement et la façon dont ces procédures sont appelées.

Au sujet de l'appel, nous dirons simplement que les interactions peuvent être gérées selon l'une des deux manières suivantes :

- asynchrone, si la procédure est appelée dynamiquement, immédiatement après la réception de l'interruption (ou dès que le programme est dans un état interruptible),

- synchrone, si le programme doit tester explicitement l'arrivée d'un évènement.

Pour la spécification des instructions à exécuter lors de la reconnaissance d'un évènement, nous considérerons quatre formes possibles de langages :

- instruction If,
- spécification de tables (GSP d'IBM, ICT d'UNIVAC),
- instruction "ON condition" de PL/1,
- diagrammes d'états.

III.1. INSTRUCTION "IF".

Après la réception d'une interruption le système recueille les informations liées à l'interruption (nom de l'élément d'image désigné, numéro de la touche fonction enfoncée, par exemple) et donne la valeur 'VRAI' à une ou plusieurs variables booléennes (par exemple LIGHTPEN ou KEY).

Par une instruction du type

```
IF LIGHTPEN THEN GOTO LABEL1;
```

ou

```
IF KEY THEN CALL ROUTINE;
```

le programmeur peut indiquer l'action à effectuer pour refléter le fait que l'évènement a été reconnu. C'est un exemple de gestion synchrone des interactions. (graphic 2)

III.2. SPECIFICATIONS DE TABLES.

L'utilisation de tables de branchement est une technique assez semblable. Cependant, elle permet de pré-spécifier les ruptures de séquence dans le déroulement du programme pour chaque type d'évènement. La méthode de pré-spécification prend souvent la forme d'un appel de sous-programme quoiqu'il soit possible de créer des ordres particuliers pour cet usage. Les paramètres d'appel sont une liste de paires ordonnées: (type de l'évènement, sous-programme de traitement spécifique).

Exemple :

CALL SETUP(Evnt(1), Routine(1), Evnt(2), Routine(2) ..., Evnt(n), Routine(n)

Seule est fixée la table de branchement et dans un système du type GSP/FORTRAN (qui ne permet pas d'opérations asynchrones), l'interruption est stockée. Dans le cours du programme, il est possible d'appeler un autre sous-programme qui testera l'arrivée d'un évènement. S'il s'en est produit un, l'appel du sous-programme attaché est généré automatiquement, sinon, le programme peut se mettre en attente ou continuer en séquence.

Cependant, dans un langage permettant l'appel asynchrone de sous-programmes (ICT "interruption control tables" d'UNIVAC, par exemple), il est possible d'utiliser cette technique pour une gestion asynchrone des interactions. Dans ce cas, le programmeur peut redéfinir dynamiquement la table de branchements.

III.3. "ON condition" de PL/1.

Il est possible d'étendre la technique utilisée dans PL/1 pour préciser les actions à effectuer quand se produit un évènement du type OVERFLOW ou division par zéro aux évènements en provenance d'une console graphique.

Exemple :

```
ON LIGHTPEN BEGIN;
  instruction 1;
      .
      .
      .
  instruction n;
      END;
```

L'exécution de l'instruction "ON" revient à modifier la table contenant l'adresse de transfert pour l'action LIGHTPEN. La technique diffère de la précédente en ce sens que l'instruction "ON" rencontrée dans une procédure est valable uniquement dans cette procédure (il y a restauration de la table lors de la sortie de la procédure). Il est utile de plus de pouvoir autoriser ou inhiber certains évènements pendant certaines séquences d'instructions. Dans ce type de traitement, la gestion des interactions est asynchrone.

III.4. DIAGRAMMES D'ETATS.

Les méthodes de spécification des interactions décrites ci-dessus obligent souvent l'utilisateur à spécifier l'état de tous les types d'évènements possibles (par exemple, si un seul type d'évènement est autorisé, l'action à effectuer doit être précisée et tous les autres évènements doivent être inhibés) et ceci doit être fait à chaque fois que la table est modifiée. De plus, un type d'évènement tel que LIGHTPEN se subdivise souvent en plusieurs évènements possibles et c'est à l'utilisateur à vérifier que l'interruption LIGHTPEN correspond bien à la désignation de l'élément qu'il attendait (cas de plusieurs menus, par exemple).

Pour pallier à ces inconvénients, W.R. Sutherland et W.M. Newman [7] ont proposé séparément une notation plus intéressante dans laquelle il est nécessaire de préciser uniquement les évènements qui doivent être reconnus et les transitions qui seront effectuées quand cet évènement est reconnu. Cette technique donne une forme de langage plus agréable à utiliser et un

code généré plus efficace. Elle permet de bien séparer la gestion des interactions des autres tâches.

Exemple : Définition d'un nuage de points et recherche du centre de gravité.

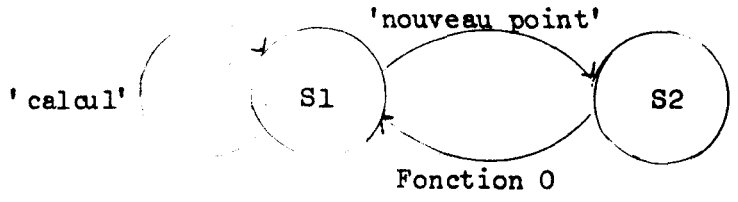


Fig. III.1

```
SYMBOL A: = Text '*';
STATE S1;
TITLE 'CHOISIR' AT <250,400>; /* Titre de l'état S1  */
ON 'NOUVEAU POINT' AT <400,300> : /* menu  */
  ENTER S2;
  ON 'CALCUL' AT  <400,200> :
  CALL GRAVITE (POINTS); /*  appel de procédure externe     */
  ENTER S1;
STATE S2;
TITLE 'AMENER LE POINT A LA POSITION DESIREE' A   <400,300>;
TRACKING WITH A; /* créé une nouvelle occurence du Symbole A
  et l'attache à la croix de tracking  */
ON FUNCTION 0 :
  INSERT CROSSCOOR TAIL POINTS /*  Ajoute les coordonnées du
  nouveau point à la liste  */
  ENTER S1;
```

C'est cette technique que nous avons utilisée pour le langage de spécification des interactions du système METAVISU, système que nous allons décrire plus complètement car les résultats de cette expérience ont servi de base à ce travail.

V. PRINCIPES DE FONCTIONNEMENT ET CRITIQUE DU SYSTEME METAVISU.

Donnons d'abord le schéma de fonctionnement du système :

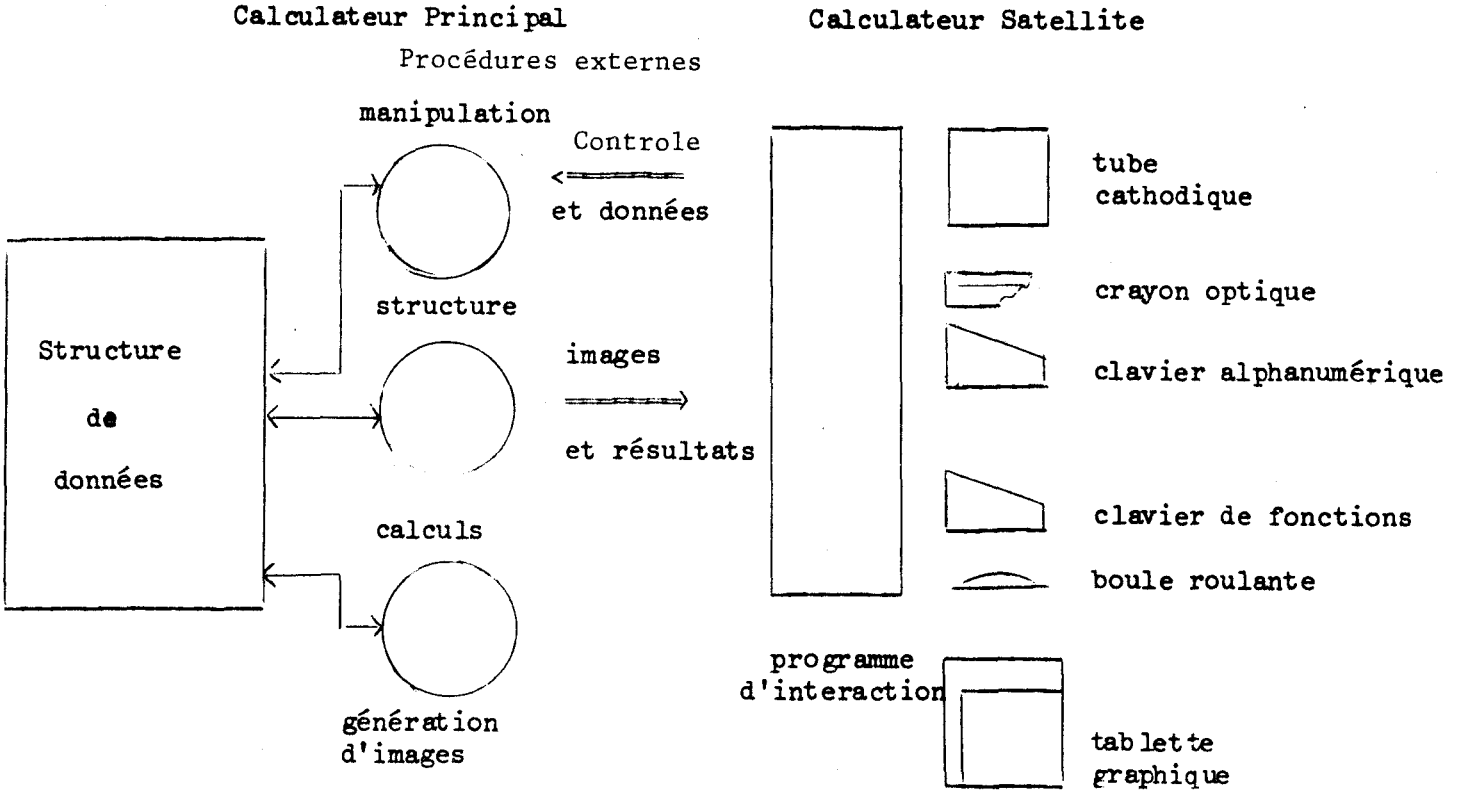


Fig. V.1

Pour permettre à l'utilisateur d'avoir une véritable activité au niveau local (calculateur satellite), en particulier, pour pouvoir construire interactivement sa structure de données, le langage de spécification des interactions permet d'effectuer certains calculs, de construire et de manipuler des "symboles", entités graphiques locales. De plus, il est possible d'utiliser des sous-programmes qui peuvent être aussi des diagrammes d'états, ce qui s'est avéré absolument indispensable, car, sans cette notion, le nombre d'états à décrire dans une application très interactive devient rapidement très élevé.

V.1. CARACTERISTIQUES PRINCIPALES DU MATERIEL ET DU SYSTEME D'EXPLOITATION UTILISES.

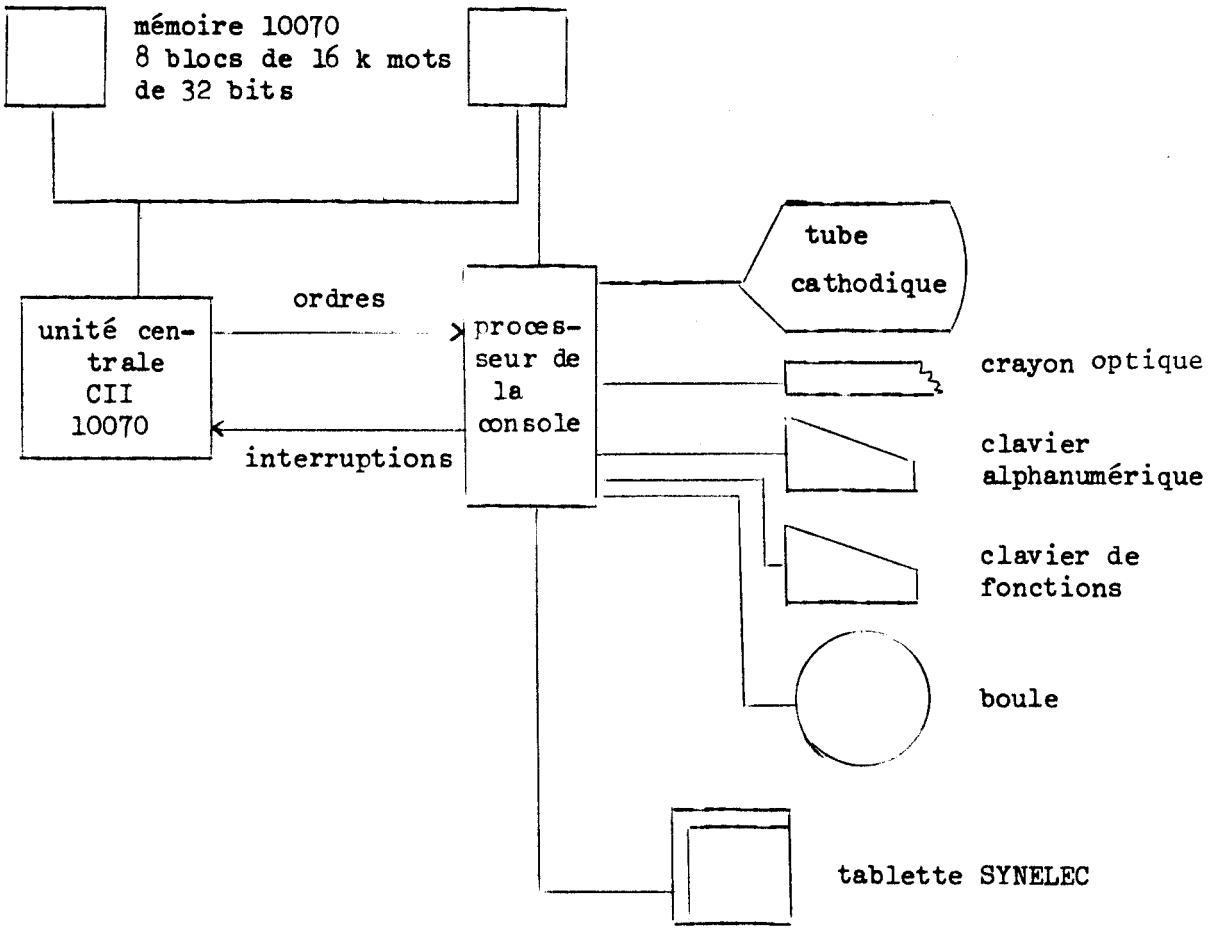


Fig. V.2

La console SINTRA VU 2000 est reliée au calculateur CII 10070 selon le mode "accès direct mémoire". La liste d'affichage est implantée dans un bloc particulier du 10070.

Le calculateur 10070 est géré par le système à partage de ressources ESOPE réalisé à l'IRIA qui utilise en particulier les notions de "mémoire virtuelle" et de "page à la demande" [19]. Le langage de génération d'images et le langage de manipulation de la structure de données associative ont été intégrés au compilateur interpréteur conversationnel CPL/1 de PL/1 réalisé

également à l'IRIA [20] . Pour remédier à l'absence de calculateur satellite, l'interpréteur du langage de spécification des interactions forme un processus moniteur particulier du système ESOPE qui travaille sur le bloc physique auquel la console accède.

Les principaux défauts du système découlent de cette configuration. Le temps de réponse du système d'exploitation est relativement mauvais car la mémoire physique abordable aux différents usagers est réduite (retrait au système du bloc mémoire accédé pour la visu et de la place occupée par le processus moniteur).

D'autre part, les procédures externes (niveau calculateur principal) écrites en CPL/1 sont interprétées ce qui augmente considérablement le temps d'exécution par rapport à un compilateur classique. Il faut noter cependant que le système a permis une gestion optimale de la structure de données associative, grâce au procédé du "couplage" (association, article de fichier, page virtuelle de l'usager).

Le schéma des transferts entre processus physiques et processus virtuels est le suivant :

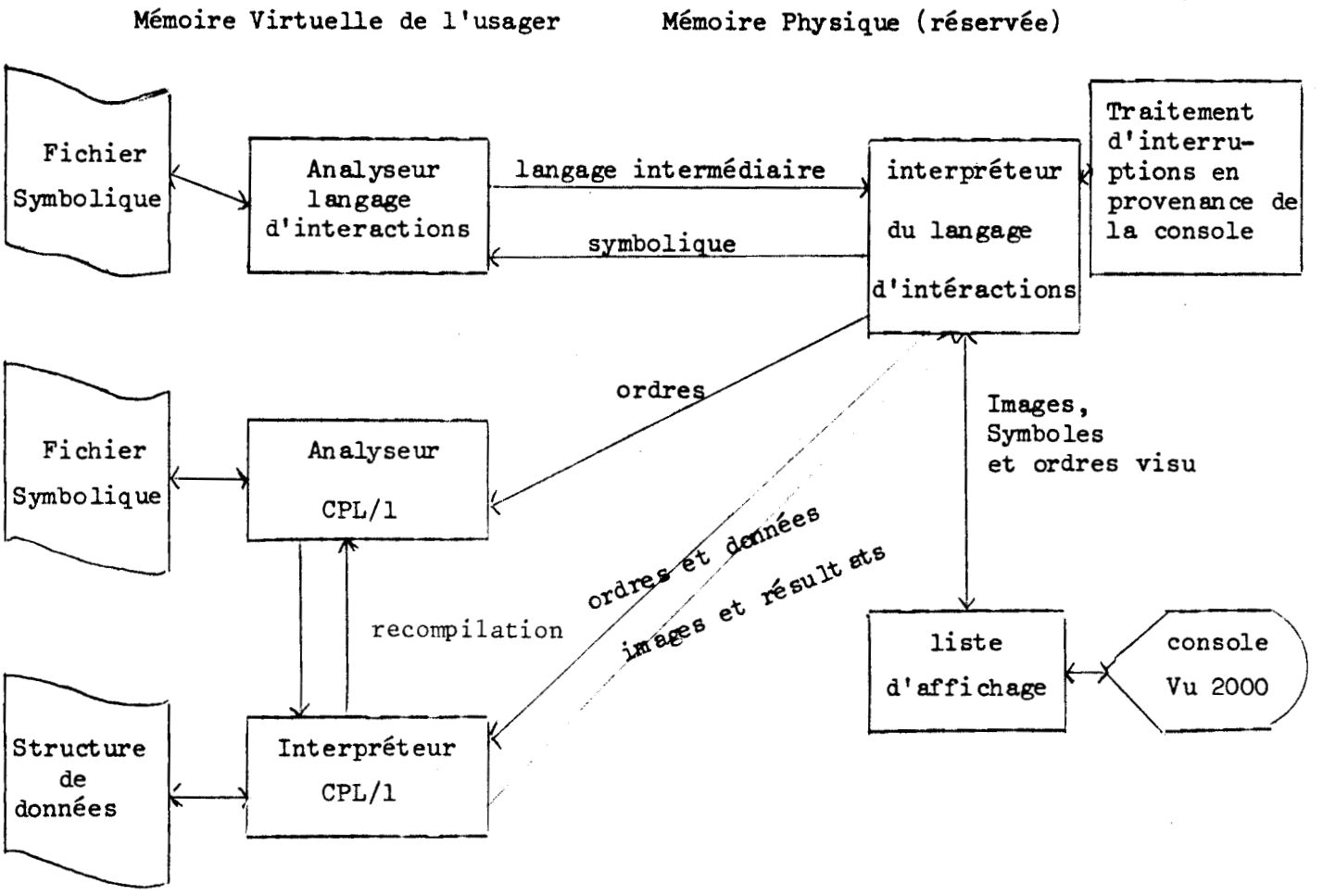


Fig. IV.3

Tous les transferts entre "mémoire physique" et "mémoire virtuelle" ont lieu au moyen du "collage" (l'adresse physique d'une page virtuelle "collée" est figée jusqu'à ce qu'elle soit libérée, la page physique correspondante n'est donc plus temporairement allouable).

V.2. CONCLUSIONS DE L'EXPERIENCE METAVISU.

Les réflexions précédentes tirées de l'expérimentation du système METAVISU ont servi de base à la définition du système.

1. Configuration hardware: 2 Calculateurs.
   - un calculateur satellite gérant les interactions,
   - un calculateur principal assurant les calculs importants et la structuration des données (éventuellement, plusieurs dans le cadre d'un réseau).

2. Indépendance totale entre le langage de spécification des interactions et les procédures externes: l'utilisateur doit pouvoir écrire ses procédures externes dans les langages qui lui conviennent.

3. Tenir compte des facilités de la console CIT Alcatel.

4. Utiliser les techniques de diagrammes d'états en la rendant un peu plus dynamique à l'exécution.

5. Générer des images décrites en trois dimensions tout en réutilisant la technique des procédures graphiques.

6. Assurer le meilleur temps de réponse possible à l'utilisateur, tout en essayant (si possible) de ménager les autres utilisateurs du calculateur principal.

VI CONFIGURATION DU SYSTEME

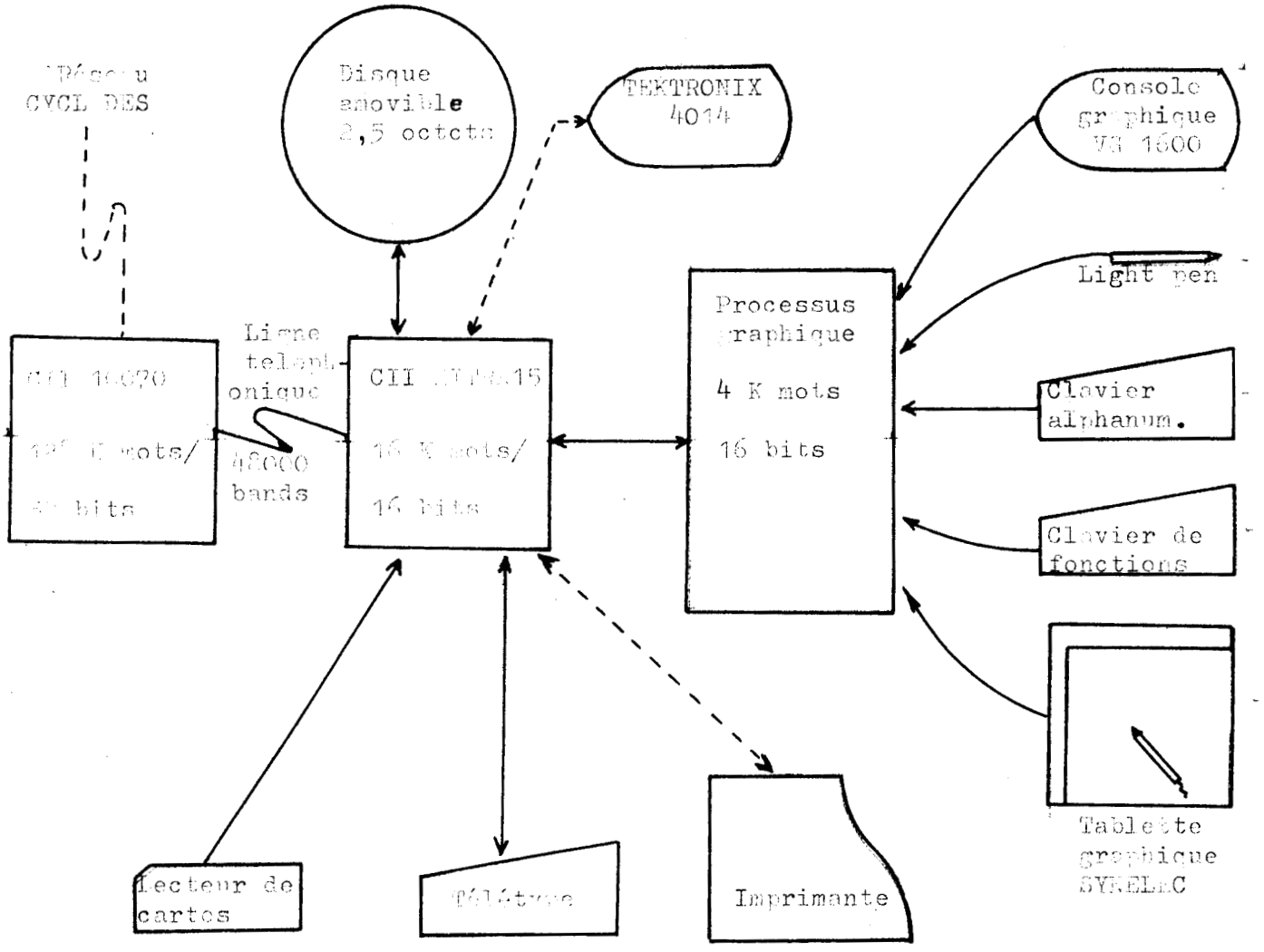


Fig. VI. 1

Les pointillés représentent des extensions prévues.
Afin de permettre le traitement d'applications relativement importantes dans le calculateur MITRA 15, seul, il est également prévu d'augmenter sa capacité mémoire.

VI.1. DESCRIPTION DE LA CONSOLE GRAPHIQUE CIT Alcatel-VG 1600.

Elle dispose d'une mémoire propre de rafraîchissement de 4k mots de 16 bits, d'un crayon optique, d'un clavier alphanumérique et d'un clavier de 32 touches fonctions. Elle est quadrichrome (rouge, orange, jaune et vert). L'écran comporte 1024×1024 points adressables. La longueur de l'incrément est de 0,25 mm.

Pour générer des images l'utilisateur dispose des instructions suivantes :

- Tracé de vecteur court (< 32 incréments)
- Tracé de vecteur long

  ces deux ordres possèdent les options suivantes :

  - absolu ou relatif,
  - continu ou pointillé,
  - allumé ou éteint,
  - 2 niveaux de brillance

- Positionnement

  sur le positionnement, on peut préciser les options suivantes :

  - surintensification, clignotement

- Changement de couleur
- Saut
- Saut avec retour (sous-programme)
- Mot vide

L'utilisateur dispose de plus des instructions de contrôle suivantes :

- Fin de cycle
- Inhibition ou Autorisation des moyens d'entrée

  (1 ordre différent pour chaque moyen d'entrée)

- Empiler une information
- Dépiler

  la console dispose d'une pile cablée à 16 niveaux dont le contenu est envoyé au calculateur à chaque désignation.

Enfin, l'utilisateur dispose des ordres suivants (ordres non mémorisés dans la liste d'affichage)

- Demande d'accès (pour effectuer une écriture)
- Demande d'information (pour effectuer une lecture)
- Début de séquence (fixe le contenu du compteur ordinal)
- Fin de Message
- Annuler Moyens d'entrée (Inhibe tous les moyens d'entrée)
- Autoriser Moyens d'entrée (restaure l'état des moyens d'entrée)
- Crayon en mode poursuite (Tracking)
- Crayon en mode désignation.

VII. LANGAGE DE SPECIFICATION DES INTERACTIONS: DESCRIPTION SYNTAXIQUE ET SEMANTIQUE.

VII.1. CONCEPTION DE DIALOGUES

Avant de décrire le langage de spécification des interactions, il est utile d'essayer de dégager les caractéristiques principales d'un bon programme interactif.

Dans ce paragraphe, nous distinguerons le programmeur d'application et l'utilisateur qui mettra en oeuvre le programme. La conception d'une application interactive pose des problèmes particuliers, car elle suppose de la part du programmeur, la connaissance de facteurs humains généralement difficiles à appréhender. Seule la double expérience d'écriture de programmes interactifs et de travail avec les utilisateurs de ces programmes permet la définition d'un bon dialogue entre l'homme et la machine, c'est-à-dire en fait d'un "langage de commande" particulièrement adapté pour l'application. Il est important que le programmeur essaie de donner à son programme les qualités suivantes :

- Simplicité du "langage de commande". Il faut que l'utilisateur puisse apprendre rapidement à mettre en oeuvre le programme.

- Clarté et cohérence du "langage de commande". Ceci permet d'éviter à l'utilisateur de nombreuses erreurs ou hésitations.

- Récupération des erreurs : l'utilisateur ne devrait jamais être pénalisé pour une entrée erronnée. Il est nécessaire d'adopter une stratégie générale pour toute l'application de récupération d'erreurs plutôt que d'essayer d'anticiper sur toutes les erreurs possibles.

- Messages d'explication. Chaque fois que c'est nécessaire, la console devrait être utilisée pour aider l'utilisateur. Une façon simple de procéder est, par exemple, d'afficher des messages expliquant à l'utilisateur ce qu'il a à faire.

L'objectif du langage de spécification des interactions est d'aider le programmeur à satisfaire ces exigences de la façon la plus simple et la plus naturelle possible.

Le langage de spécification des interactions peut être divisé en deux parties, une partie manipulation de données proche des langages de programmation habituels, une partie description de dialogues interactifs donnant au programme, une structure de diagrammes d'états.

VII.2. PRELIMINAIRES.

Notations Syntaxiques : La notation utilisée pour la description de la syntaxe est la notation usuelle de Backus-Naur. Les mots-clés du langage seront écrits en caractères majuscules. ∧ dénote la chaîne vide.

De manière à faciliter la lecture des formules syntaxiques, les différentes alternatives d'une définition sont représentées sur des lignes successives commençant par le caractère métalinguistique |. De ce fait le caractère spécial | apparaissant ailleurs qu'en début de ligne peut être aisément interprété comme un mot-clé et non comme un signe métalinguistique.

Identificateurs : un identificateur est composé d'une succession d'au plus six caractères alphanumériques dont le premier est alphabétique.

Chaînes de caractères : Les chaînes de caractères sont formées d'une succession de caractères EBCDIC (à l'exclusion de l'apostrophe) comprise entre deux apostrophes

'CECI _ EST _ UNE _ CHAINE'

L'apostrophe à l'intérieur d'une chaîne sera représentée par le caractère spécial " (guillemet). La longueur maximale d'une chaîne est de deux cent cinquante cinq caractères.

Commentaires : Un commentaire est une suite quelconque de caractères EBCDIC quelconques (ne comprenant pas la suite /*) comprise à l'intérieur des symboles /* et */.

Exemple :

/* CECI EST UN COMMENTAIRE */

Un commentaire est équivalent à un espace et peut apparaître entre deux éléments lexicaux quelconques.

Signification des espaces : Un ou plusieurs espaces doivent normalement figurer entre deux éléments lexicaux successifs. Cet espacement est cependant facultatif si l'un des éléments lexicaux successifs est formé de symboles spéciaux.

VII.3. ELEMENTS DU LANGAGE ET DECLARATIONS.

Les principaux éléments du langage sont les variables, les tableaux, les listes et les procédures. Chaque élément utilisé doit faire l'objet d'une déclaration. Celle-ci introduit le nom de l'élément et associe un type à ce nom. La notion de type permet de donner une grande clarté au programme. Elle rend possible la détection de certaines erreurs dès la compilation. Nous verrons en effet que les instructions et les expressions doivent être formées d'éléments dont les types sont cohérents. Le langage comprend sept types simples : INTEGER, REAL, PAIR, STRING, ITEM, SYMBOL et FRAME et permet d'utiliser des types structurés comme ARRAY ou LIST.

Syntaxe :

7.3.1 <déclaration> ::= <déclaration simple>
| <déclaration de tableau>
| <déclaration de liste>
| <déclaration de procédure>;

7.3.2. <déclaration simple> ::= <type> <liste d'identificateurs>;

7.3.3 <liste d'identificateurs> ::= <identificateur>
| <identificateur>, <liste d'identificateurs>;

7.3.4. <type> ::= INTEGER
| REAL
| PAIR
| STRING
| ITEM
| SYMBOL
| FRAME;

7.3.5. <déclaration de tableau> ::= <type> ARRAY <liste de tableaux>;

7.3.6. <liste de tableaux> ::= <tableau>
| <tableau>, <liste de tableaux>;

7.3.7. <tableau> ::= <identificateur> [<liste de paires de borne>] ;

7.3.8. <liste de paires de bornes> ::= <paire de bornes>
| <paire de bornes> ,<liste de paires de bornes>;

7.3.9. <paire de bornes> ::= <entier> : <entier>;

7.3.10. <déclaration de liste> ::= <type> LIST <liste d'identificateurs>;

7.3.11. <déclaration de procédure> ::= <déclaration de procédure interne>
| <déclaration de procédure externe>
| <déclaration de procédure en référence>;

7.3.12. <déclaration de procédure interne> ::= <type de procédure> PROCEDURE
<identificateur> <partie paramètres formels>';'<corps de procédure>;

7.3.13. <type de procedure> :: = Λ
| <type>
| <type> LIST;

7.3.14. <Partie paramètres formels> ::= Λ
| (<liste d'identificateurs>)'; <liste de déclarations de paramètres formels>;

7.3.15. <liste de déclarations de paramètres formels> ::= <déclaration de paramètres formels>
| <déclaration de paramètres formels> ; <liste de déclarations de paramètres formels>;

7.3.16. <déclaration de paramètres formels> ::= <déclaration simple>
| <déclaration de liste>
| <déclaration de tableau formel>;

7.3.17. <déclaration de tableau formel> ::= <type> ARRAY <liste d'identificateurs>;

7.3.18. <déclaration de procédure en référence> :: = REF <type de procedure> PROCEDURE <identificateur> <spécification de paramètres formels>;

7.3.19. <spécification de paramètres formels> ::= Λ
| (<liste de paramètres formels>);

7.3.20. <liste de paramètres formels> ::= <paramètre formel>
| <paramètre formel>, <liste de paramètres formels>;

7.3.21. <Paramètre formel> ::= <type>
| <type> ARRAY
| <type> LIST;

7.3.22. <déclaration de procédure externe> ::= EXTERNAL PROCEDURE <identificateur> <spécification de paramètres formels>;

Exemples :

7.3.3. INTEGER A,B
PAIR E1

7.3.5. REAL ARRAY D [1:10,3:15]

7.3.10 FRAME LIST F

7.3.12. PAIR PROCEDURE G(X,Y);
INTEGER X,Y;
BEGIN
RETURN <X,Y>;
END

7.3.18. REF PROCEDURE ZX(INTEGER,REAL ARRAY,SYMBOL)

7.3.22. EXTERNAL PROCEDURE FX(ITEM LIST,REAL)

Sémantique :

a/ types simples
- INTEGER : un entier est compris entre -2^{16} et $2^{16}-1$;
- REAL
- PAIR : le type couple peut être défini de la façon suivante : MODE PAIR = STRUCT (INT FIRST,INT SECOND) en utilisant la notation ALGOL 68. Il sert essentiellement à la manipulation de coordonnées.
- STRING : les chaînes de caractères sont limitées à 255 caractères.
- ITEM ou associateur : ce type d'élément sert essentiellement à associer un élément de structure de données (externe au satellite) à une partie de l'image.
- SYMBOL : entité graphique construite en langage de spécification des interactions. Sert essentiellement à l'édition graphique

- FRAME : entité graphique construite par une procédure externe.

b/ Types structurés
- ARRAY : ce sont des tableaux statiques classiques.
- LIST : les listes sont linéaires et composées d'éléments homogènes. Il n'existe pas de possibilité de sous-listes. Elles sont indispensables pour un certain nombre d'opérations interactives (exemple: suivi d'un tracé sur la tablette) car elles sont plus souples que les tableaux (possibilités de mise à jour par insertion, retrait d'un élément quelconque).

c/ Procédures

Nous distinguons en fait deux types de procédures :

- les procédures s'exécutant dans le calculateur satellite et qui sont généralement écrites en langage de spécification des interactions. Elles peuvent être compilées à part (option REF). Ces procédures peuvent être des fonctions et dans ce cas leur déclaration est précédée du type du résultat.

- les procédures externes s'exécutant dans un autre calculateur et écrites dans des langages divers.

Remarque : les paramètres des procédures externes ne peuvent pas être du type SYMBOL (ou d'un type structuré construit à partir de ce type).

VII.4. EXPRESSIONS.

Nous distinguerons suivant les types de variables, les différentes expressions possibles.

a/ Définitions communes.

syntaxe :

7.4.1.a <variable> ::= <identificateur>
| <identificateur> [<suite d'indices>] ;

7.4.2.a <suites d'indices> ::= <expression arithmétique>
| <expression arithmétique>,<suite d'indices>;

7.4.3.a <expression> ::= <expression arithmétique>
| <expression de couple>
| <expression de chaîne>
| <expression de symbole>
| <expression d'item>
| <expression de liste> ;

7.4.4.a <appel de fonction> ::= <identificateur de fonction>
<identificateur de fonction> (<liste de paramètres effectifs>);

7.4.5.a <liste de paramètres effectifs> ::= <paramètre effectif>
| <paramètre effectif>, <liste de paramètres effectifs>;

7.4.6.a <paramètre effectif> ::= expression

Exemples :

7.4.1. a A
B
B [1,I+4]

7.4.4. a F
F(X,I+J)

Une fonction est soit une procédure qui a été déclarée avec un type, soit une fonction standard fournie par le système et qui ne doit donc pas être redéclarée.

(voir le paragraphe VII.8 pour la liste des fonctions et procédures standards).

b/ Expressions arithmétiques :

Elles portent sur les variables de type INTEGER ou REAL.

Syntaxe :

7.4.1.b <expression arithmétique> ::= <terme>
| <expression arithmétique> <opérateur additif> <terme>
<opérateur additif> <terme>;

7.4.2.b <opérateur additif> ::= +
|-;

7.4.3.b <terme> ::= <facteur>
| <terme> <opérateur multiplicatif> <facteur>;

7.4.4.b <opérateur multiplicatif> ::= *
| /;

7.4.5.b. <facteur> ::= <primaire>
| <facteur> ** <primaire>;

7.4.6.b <primaire> ::= <constante arithmétique >
| <variable arithmétique>
| <appel de fonction>
| (<expression arithmétique>);

7.4.7.b <constante arithmétique> ::= <entier sans signe>
| <réel sans signe>;

7.4.8.b <variable arithmétique > ::= <variable>
| <sélecteur de champ> <variable>;

7.4.9.b <sélecteur de champ> ::= FIRST
| SECOND;

Exemples :

```
A + B [I]
A * (B+C)
(A - F(X,Y)/FIRST D)**4
```

Sémantique : les sélecteurs de champ portent sur des variables de type couple.

Priorité des opérateurs

1 ** (exponentiation)
2 * et /
3 + et -

c/ Expressions de couple : les couples servent essentiellement à la manipulation de coordonnées écran.

Syntaxe :

7.4.1.c <expression de couple> ::= <terme de couple>
| <opérateur additif> <terme de couple>
| <extension de couple> <opérateur additif> <terme de couple>;

7.4.2.c <terme de couple> ::= <facteur de couple>
| <facteur de couple> <opérateur multiplicatif> <primaire>;

7.4.3.c <facteur de couple> ::= <variable>
| <appel de fonction>
| (<expression de couple>)
| '<' <expression arithmétique>, <expression arithmétique> '>';

Les suites de caractères '<' , '>' indiquent les caractères < et > ne sont pas des signes métalinguistiques mais des éléments du langage.

Exemples :
A + B
(A + <C+4,D>) * E

Sémantique : le primaire a été défini par la règle 7.4.6.b sur les expressions arithmétiques.

Priorité des opérateurs :
1 * et /
2 + et -
Nous représenterons un couple par <a,b> où :
a est la valeur du premier élément du couple,
b est la valeur du second élément du couple
+ <a,b> = <a,b>
- <a,b> = <-a,-b>
<a,b> + <c,d> = <a+c,b+d>
<a,b> - <c,d> = <a-c,b-d>
<a,b> * c = <a*c,b*c> où c est un entier
<a,b>/c = <a/c,b/c>

d/ Expression de chaîne :
Syntaxe :
7.4.1.d <expression de chaîne> ::= <terme de chaîne>
| <expression de chaîne> || <terme de chaîne>;

7.4.2.d <terme de chaîne> ::= <variable>
| <appel de fonction>
| <chaîne>;

Exemples :

'ABC'

A || B

A || 'AB' || F(H)

Sémantique :

|| est l'opération de concaténation

'AB' || 'CD' ⟶ 'ABCD'

e/ Expressions de symbôle : permettent de construire des entités graphiques locales.

Syntaxe :

7.4.1.e <expression de symbole> ::= <terme de symbôle>
| <expression de symbôle> <terme de symbôle>
| NULL;

7.4.2.e <terme de symbôle> ::= LINE <ligne>
| MOVE <déplacement>
| TEXT <texte>
| <variable>
| <appel de fonction>
| HITSYMBOL;

7.4.3.e <ligne> ::= <déplacement de ligne>
| WITH <expression arithmétique> <ligne>
| <déplacement de ligne> <ligne>;

7.4.4.e <déplacement de ligne> ::= <déplacement>
| FROM <expression de couple> <déplacement>;

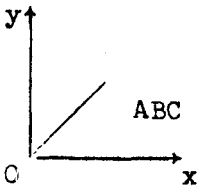
7.4.5.e <déplacement> ::= OF <expression de couple>
| TO <expression de couple>;

7.4.6.e <texte> ::= <expression de chaîne>
| WITH <expression arithmétique> <expression de chaîne>

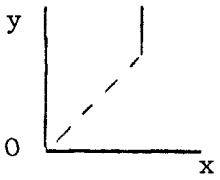
Exemples :

LINE OF <10,10> MOVE TO <15,5> TEXT 'ABC'

ce qui donne le symbole suivant :

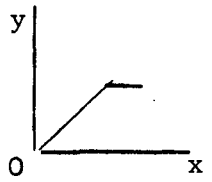


LINE WITH 2 TO <15,15> WITH 1 OF <0,5>

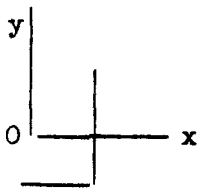


HITSYMBOL A MOVE TO <0,0> B

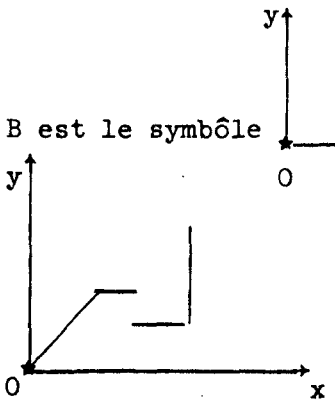
où le Hitsymbol est le symbole :



A est le symbole :



(move to <10,10> line
to <10,-5> line
< 0,-5>



Sémantique :

Les symbôles sont décrits dans leur espace propre. Les couples intervenant dans les lignes et les déplacements sont en incréments écran (0,25 mm).

HITSYMBOL est le dernier symbôle désigné (voir paragraphe VII.6).

NULL est le symbôle vide.

Caractéristiques de trait :

L'expression arithmétique suivant le WITH est interprétée de la façon suivante :

e = 1 continu
e = 2 pointillé
e = 3 continu sous brillant
e = 4 pointillé sous brillant

Toute autre valeur est équivalente à e = 1 (option par défaut). L'option WITH est valable jusqu'à l'option WITH suivante ou le mot-clé LINE suivant.

Caractéristiques de texte :

L'expression arithmétique suivant le WITH est interprétée de la façon suivante :

e = 1 taille 1 (moyenne 1,5×1,25 mm par caractère)
e = 2 taille 2 (moyenne 3×2,5 mm par caractère)
e = 3 taille 3 (moyenne 4,5×3,75 mm par caractère)
e = 4 taille 4 (moyenne 6×5 mm par caractère)

Toute autre valeur est équivalente à e = 2 (option par défaut).

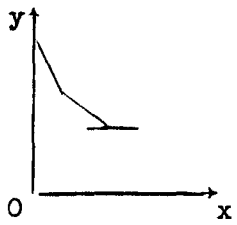
Déplacements et lignes :

OF indique un déplacement en relatif par rapport à la position atteinte.
TO indique un déplacement par rapport à l'origine du symbôle.

Option FROM :

Exemple :

LINE FROM <10,10> TO <5,15> OF <-5,10> FROM < 5,10> OF <10,0>



cette expression est équivalente à

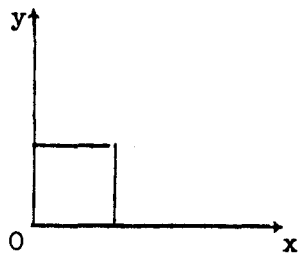
MOVE TO <10,10> LINE TO <5,15> OF <-5,10> MOVE TO <5,10> LINE OF <10,0>

Déplacement introduit par un symbôle :

Les symbôles ne sont pas neutres, c'est-à-dire qu'ils introduisent un déplacement.

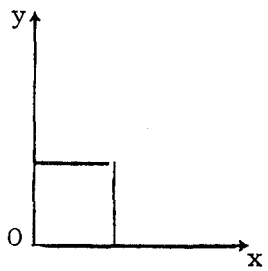
Exemple :

Soit A le symbôle décrit par :



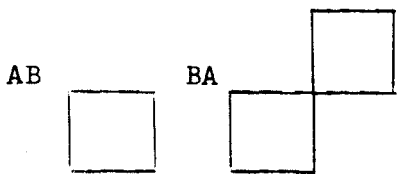
LINE TO <10,0> TO <10,10> TO <0,10> TO <0,0>

et B le symbôle



LINE TO <10,0> TO <10,10> LINE FROM <0,0> TO <0,10> TO <10,10>

Ces deux symbôles ne sont pas identiques car le symbôle obtenu en concaténant A et B n'est pas le même que celui obtenu en concaténant B et A.



f/ Expressions d'items :

Syntaxe :

7.4.1.f <expression d'item> ::= <variable>
| <appel de fonction>;

Sémantique :

Un item est une référence à un élément d'une structure de données externe associé à une partie d'image. Il n'est donc pas possible d'effectuer des opérations sur ce type d'élément.

g/ Expressions de liste :

Syntaxe :

7.4.1.g <expression de liste> ::= <liste>
| <expression de liste> || <liste>;

7.4.2.g <liste> ::= <identificateur>
| <appel de fonction>

Exemples :

L || L1
F(G)

Sémantique : Concaténation de deux listes. Les éléments composant ces listes doivent être de même type.

VI.5. INSTRUCTIONS.

a/ Instruction d'affectation :

Syntaxe :

7.5.1.a

<instruction d'affectation> ::= <variable arithmétique> ::= <expression> ;

Exemples :

A:=B
FIRST C:=(A+B)*D
A [4,I] :=5+F(X)
S [4] :=LINE TO <10,10> A

Sémantique : Il doit y avoir conformité de type entre la variable et l'expression.

b/ Conditions.

Syntaxe :

7.5.1.b <condition> ::= <facteur booléen>
|<condition> <opérateur booléen> <facteur booléen>,

7.5.2.b <facteur booléen> ::= <relation>
| ¬ <relation>;

7.5.3.b <relation> ::= <expression> <opérateur de relation> <expression>
| (<condition>);

7.5.4.b <opérateur booléen> ::= &
|'|';

7.5.5.b <opérateur de relation> ::= '>'
|'<'
| > =
| < =
| =
| ¬ =;

Exemples :

A > B & (A=C | D=F)
A ¬ = B
¬ (A=B | B> =C)

Sémantique :

& est le "et" logique.
| est le ou inclusif
¬ est la négation

les opérateurs <, <=, > et >= portent uniquement sur les expressions arithmétiques.

cas de = et ¬ =

on peut tester :

<expression arithmétique> { =, ¬= } <expression arithmétique>

<expression de couple> { =, ¬= } <expression de couple>

<expression de chaîne> { =, ¬= } <expression de chaîne>

<expression d'item> { =, ¬= } <expression d'item>

<expression d'item> { =, ¬= } <expression arithmétique>

<expression arithmétique> { =, ¬= } <expression d'item>

c/ Instruction Conditionnelle.

Syntaxe :

7.5.1.c <instruction conditionnelle> ::= IF <condition> THEN <instruction inconditionnelle> | IF <condition> THEN <instruction inconditionnelle> ELSE <instruction>;

Exemples :

IF A > B THEN A:=B ELSE A:=C

IF A = B THEN A:=1

Sémantique : Si la condition est vérifiée, l'instruction suivant le THEN est exécutée sinon on exécute la partie ELSE.

d/ Boucles

Syntaxe :

7.5.1.d <boucle> ::= <proposition FOR>
|<proposition WHILE>;

7.5.2.d <proposition FOR> ::= FOR <variable arithmétique> :=
<expression arithmétique> <suite du FOR>;

7.5.3.d <suite du FOR> ::= STEP <expression arithmétique> UNTIL
<expression arithmétique> <partie WHILE>
|UNTIL <expression arithmétique> <partie WHILE>;

7.5.4.d <partie WHILE> ::= DO <instruction>
|<proposition WHILE>;

7.5.5.d <proposition WHILE> ::= WHILE <condition> DO <instruction>

Exemples :

```
FOR I:=1 STEP 2 UNTIL IN DO A:= A:=A+1
FOR I:=1 UNTIL N DO A:= B+I
FOR J[I]:=1 STEP N + A UNTIL K WHILE A > 0 DO A:=A-J[I]
WHILE FIRST C > 0 DO C:=C - <A,B>
```

Sémantique :

FOR : le pas et la limite de la boucle sont évalués à la première itération. Lorsque le pas est omis, celui-ci est pris implicitement à 1

WHILE : tant que la condition est vérifiée, on exécute l'instruction.

e/ Instruction CASE:

Syntaxe :

7.5.1.c <instruction CASE> ::= CASE <variable arithmétique>
OF <instruction> <suite de cas>;

7.5.2.c <suite de cas> ::= OR <instruction>
|OR <instruction> <suite de cas>;

Exemples :

```
CASE A
OF A:=B /* CAS A = 1 */
OR A:=C /* cas A = 2 */
OR A:=D /* cas A = 3 */
```

Sémantique : Suivant la valeur de la variable, l'instruction correspondante sera effectuée. Si la variable possède une valeur différente (< 1 ou > nombre de cas), aucune des instructions spécifiées dans le CASE ne sera exécutée.

f/ Instructions particulières sur les listes.

Syntaxe :

7.5.1.f <instruction de liste> :: = <insertion>
|<retrait>;

7.5.2.f <insertion> ::= INSERT <expression> <fin d'insertion>;

7.5.3.f <fin d'insertion> ::= IN <identificateur> BEFORE <expression arithmétique> | TAIL <identificateur >
| HEAD <identificateur>;

7.5.4.f <retrait> ::= REMOVE <expression arithmétique> FROM <identificateur> | REMOVE TAIL <identificateur>
|REMOVE HEAD <identificateur>;

Exemples :

```
INSERT A+4 /* élément à insérer */ IN L /* Liste d'entiers */
BEFORE 5 /* indice de l'élément devant lequel on insère */
INSERT HITSYMBOL TAIL LS /* ajoute un élément en fin de la liste */
INSERT  <A,B>  HEAD LC /* ajoute un élément en tête de liste  */
REMOVE S /* indice de l'élément à retirer */ FROM L
REMOVE TAIL LS /* retrait du dernier élément  */
REMOVE HEAD LS /* retrait du premier élément  */
```

Sémantique : Les listes étant homogènes, le type de l'élément à insérer et de la liste doivent être identiques. La tentative d'insertion devant un élément inexistant provoque une erreur. La tentative de retrait d'un élément inexistant n'est pas effectuée.

g/ Instructions sur les symbôles : permettent la manipulation des symbôles en particulier l'affichage.

Syntaxe :

7.5.1.g <instruction symbole> ::= <copy>
| <move>
| <light>
| <delete>;

7.5.2.g <variable de symbôle> ::= <variable>
| HITSYMBOL;

7.5.3.g <copy> ::= COPY <variable de symbôle> AT <expression de couple>
<partie light>

7.5.4.g <partie light> ::= Λ
| LIGHT <expression arithmétique>, <expression arithmétique> ,
<expression arithmétique>

7.5.5 .g <partie symbôle> ::= <variable de symbôle>
| HITSYMBOC ;

7.5.6.g <move> ::= MOVE <partie symbôle> OF <expression de couple>
| MOVE <partie symbôle> TO <expression de couple>;

7.5.7.g <light> ::= LIGHT <partie symbôle>, <expression arithmétique>,
<expression arithmétique>,<expression arithmétique >;

7.5.8.g <delete> ::= DELETE <partie symbôle>;

Exemples :

```
COPY A AT <500,100> LIGHT 0,1,2
COPY HITSYMBOL AT <10,10>
MOVE HITSYMBOC TO <0,0>
LIGHT A,1,2,3
DELETE A
DELETE HITSYMBOC
```

Sémantique : Nous avons vu comment on pouvait décrire un symbôle. La description d'un symbôle se fait dans son espace propre et n'entraîne pas son apparition sur l'écran. Pour le faire apparaître il faut créer une occurence par l'instruction COPY en précisant le positionnement sur l'écran et les caractéristiques graphiques de l'occurence (intensité, clignotement, couleur) par l'option LIGHT.

- intensité = 0	normale
intensité = 1	surintensifiée
- clignotement = 0	normal
clignotement = 1	l'occurence clignote
- couleur = 1	rouge
= 2	orange
= 3	jaune
= 4	vert

Les options par défaut sont : intensité = 0, clignotement = 0, couleur = 4. Un symbôle peut avoir un nombre quelconque d'occurences.

Une occurence n'est accessible indépendamment des autres occurences du même symbol si elle a été préalablement désignée. Elle devient alors le HITSYMBOC (voir paragraphe VII.6.

L'utilisation du mot-clé HITSYMBOL (repérant le dernier symbôle désigné) est équivalente à l'emploi du nom du symbôle désigné.

L'instruction MOVE permet de déplacer sur l'écran les occurences d'un symbôle soit d'une certaine valeur (option OF) soit jusqu'à une certaine position (option TO). Si le mot-clé HITSYMBOC est utilisé, seule la dernière occurence désignée est déplacée.

L'instruction LIGHT permet de modifier les caractéristiques graphiques des occurences d'un symbôle. Si la valeur de l'expression arithmétique donnant la caractéristique est différente des valeurs précisées précédemment, cette caractéristique n'est pas modifiée.

L'instruction DELETE permet de supprimer les occurences d'un symbôle.

Remarques : La réaffectation d'un symbôle supprime toutes les occurences de ce symbôle. Si un symbôle ne possède pas d'occurence, les instructions MOVE, LIGHT et DELETE sont sans effet.

h/ Instruction sur les images : permettent d'afficher et de manipuler les images.

Syntaxe :

7.5.1.h <instruction image> ::= LIGHT <variable d'image>, <expression arithmétique >, <expression arithmétique>, <expression arithmétique>
| DISPLAY <variable d'image>
| DELETE <variable d'image>;

7.5.2.h <variable d'image> ::= <identificateur>
| <identificateur> [<suite d'indices>]
| HITFRAME;

Exemples :

```
DISPLAY A
LIGHT HITFRAME, 0,1,3
```

Sémantique : Les images sont générées par les procédures externes. Les seules manipulations possibles sont :

- afficher l'image (DISPLAY): on transfère l'image depuis le calculateur satellite vers la console.

- supprimer l'image de l'écran (DELETE)

- modifier les caractéristiques graphiques de l'image (LIGHT).Les paramètres de l'instruction LIGHT ont la même signification que dans l'instruction LIGHT sur les symbôles.

Le HITFRAME est la dernière partie d'image désignée (voir paragraphe VII.6 pour la définition précise de ce terme). Il est manipulable comme une image.

Remarque : L'instruction DISPLAY sur une image déjà affichée n'est pas inopérante: elle permet de revenir à l'état initial de l'image (elle rétablit les caractéristiques graphiques et réaffiche éventuellement le HITFRAME).

i/ Appel de procédure:

Syntaxe :

7.5.1.i <appel de procédure> ::= <appel de procédure interne>
| <appel de procédure externe>;

7.5.2.i <appel de procédure interne> ::= <identificateur>
| <identificateur> (<liste de paramètres effectifs>);

7.5.3.i <appel de procédure externe> ::= CALL <identificateur>
| CALL <identificateur> (<liste de paramètres effectifs de procédure externe>) ;

7.5.4.i <liste de paramètres effectifs de procédure externe> ::=
<paramètre effectif de procédure externe>
| <paramètre effectif de procédure externe>,<liste de paramètres effectifs de procédures externes>;

7.5.5.i <paramètre effectif de procédure externe> ::= <expression>
| <ordre> <variable externe>;

7.5.6.i <ordre> ::= ^
| *
| $;

7.5.7.i <variable externe> ::= <variable>
| <identificateur> [<liste de paires de bornes de variable externe>];

7.5.8.i <liste de paires de bornes de variable externe> ::=
<paire de bornes externe>
| <paire de borne externe>, <liste de paires de bornes de variable externe>;

7.5.9.i <paire de bornes externe> ::= <expression arithmétique> :
<expression arithmétique>;

Exemples :

A(X,Y,LINE TO <X,Y>).

CALL EXT (A,*B [1:A,3:5] , F(X,Y), $L [I] , *F1 [1:5])

Sémantique :

- Procédures internes.

La passation des paramètres se fait par référence avec éventuellement création de dummy dans le cas de la passation d'une expression.

Remarque : Les mêmes règles sont valables pour les appels de fonction.

- Procédures externes.

Ordre Λ transfert du paramètre dans le sens langage d'interaction

Procédure externe. Cet ordre est évidemment le seul permis pour les expressions.

*le paramètre est transféré depuis les procédures externes vers le langage d'interaction (au retour de la procédure externe)

$ le transfert s'effectue dans les deux sens.

Remarques: les expressions figurant en indice dans un appel de procédure externe sont évaluées à l'appel de la procédure et ne sont pas modifiées par un éventuel reflet.

Soit par exemple, l'instruction

CALL F($ A,*B [3:A])

avec A = 5 à l'appel; cette instruction est équivalente à

CALL F($A,*B [3:5])

les symbôles ne peuvent pas être passés en paramètre de procédure externe.

les images passées en paramètres sont nécessairement précédées de l'ordre * car elles ne peuvent être envoyées que par les procédures externes.

Enfin, il doit y avoir identité de type entre les paramètres formels spécifiés lors de la déclaration de la procédure (interne ou externe) et les paramètres effectifs à l'appel de la procédure.

j/ Retour de procédure.

Ceci s'applique bien évidemment aux procédures internes.

Syntaxe :

7.5.1.j <retour de procédure> ::= RETURN
| RETURN <expression>;

Exemples :

RETURN
RETURN LINE TO <X,Y> A
RETURN 'ABC'

Sémantique : Cet ordre indique que l'exécution de la procédure est terminée et qu'on revient au programme appelant. Une expression est autorisée dans le cas où la procédure a été déclarée avec un type (fonction). Dans ce cas le type de l'expression et le type de la fonction doivent être identiques.

k/ Instructions de mode : Instructions spéciales ne pouvant pas figurer que dans des zones précises du programme et qui permettent de spécifier le mode de travail de la console.

Syntaxe :

7.5.1.k <instruction de mode> ::= <instruction titre>
| <instruction de tracking>
| <instruction d'édition>;

7.5.2.k <instruction titre> ::= TITLE <expression de symbôle>
<suite titre>;

7.5.3.k <suite titre> ::= <partie light>
| AT <expression de couple> <partie light>;

7.5.4.k <instruction de tracking> ::= TRACKING <suite tracking>
<rubberband>

7.5.5.k <suite tracking> ::= Λ
| AT <expression de couple>
| WITH HITSYMBOC
| WITH <variable de symbôle> AT <expression de couple>
<partie light> ;

7.5.6.k <rubberband> ::= RUBBERBAND
| RUBBERBAND FROM <expression de couple>;

7.5.7.k <instruction d'édition> ::= EDIT
| EDIT AT <expression de couple>
| EDIT <expression de chaîne>
EDIT <expression de chaîne> AT <expression de couple>;

Exemples :

```
TITLE TEXT 'ENFONCER LA TOUCHE NO 2'
TITLE TEXT 'DESIGNER LE' MOVE OF <5,0> LINE OF
<5,0> OF <0,5> OF <-5,0> OF <0,-5> AT <200,500>
LIGHT  0,0,2 ;
```

ce qui donne sur l'écran à la position 200,500

DESIGNER LE  (de couleur orange)

```
TRACKING
TRACKING WITH S AT <200,300>
TRACKING WITH HITSYMBOC RUBBERBAND
TRACKING RUBBERBAND FROM <10,0>
TRACKING AT <0,0>
EDIT
EDIT 'TEXTE INITIAL' AT <-200,300>
```

Sémantique :

Nous verrons dans le paragraphe VII.6 les règles d'utilisation de ces instructions.

- instruction titre: son rôle est essentiellement de permettre l'affichage de messages destinés à aider l'utilisateur dans la mise en oeuvre du programme. Le programmeur peut soit positionner lui-même ses titres, soit laisser au système le soin de s'en charger. Le choix d'une expression de symbôle plutôt qu'une expression de chaîne dans un titre permet au programmeur davantage de souplesse dans le titre.

- instruction de tracking: Elle permet le déplacement d'une croix avec le crayon. En fait, elle précise que l'utilisateur fera du tracking et donne les options initiales du tracking.

Il est possible d'attacher une occurrence d'un symbôle à la croix c'est-à-dire que le positionnement de l'occurrence sera le même et variera avec le positionnement de la croix.

Deux options sont possibles :
- soit utiliser la dernière occurrence désignée (WITH HITSYMBOC),
- soit créer une nouvelle occurrence qui sera attachée à la croix (WITH <variable de symbole>)

- l'instruction EDIT permet d'éditer une chaîne de caractères sur la console. Il est possible de modifier une chaîne initiale ou d'en créer une nouvelle.

ℓ/ Instructions de branchement vers un état :

Syntaxe :

7.5.1.ℓ <instruction de branchement> ::= ENTER <identificateur>
|IDEM;

Exemples :

ENTER S1
IDEM

Sémantique : L'identificateur est un nom d'état (voir § VII.6)

m/ Bloc d'instructions :

Syntaxe :

7.5.1.m <bloc d'instructions> ::= BEGIN <suite d'instructions> END

7.5.2.m <suite d'instructions> ::= Λ
| <instruction>
| <instruction> ';' <suite d'instructions>;

7.5.3.m <instruction> ::= <instruction conditionnelle>
| <instruction inconditionnelle>;

7.5.4.m <instruction inconditionnelle> ::= <instruction d'affectation>
| <instruction CASE>
| <instruction de boucle>
| <instruction sur les listes>
| <instruction sur les symbôles>
| <instruction sur les images>
| <appel de procédure>
| <instruction de mode>
| <instruction de branchement>
| <bloc d'instructions>
| EXIT ;

Sémantique : l'instruction EXIT termine le programme d'interactions et rend la main au moniteur d'interactions.

VII.6. ETATS ET EVENEMENTS.

Syntaxe :

7.6.1. <état> ::= STATE <identificateur> ';' <zone mode> <description d'évènements> ENDSTATE;

7.6.2. <zone mode> :: =<suite d'instructions>;

7.6.3. <description d'évènements> ::= ON <évènement> : <suite d'instructions>
| ON <évènement>:<suite d'instructions>';'<description d'évènements>

7.6.4. <évènement> ::= KEY <caractère>
| FUNCTION <entier>
| PEN <variable>
| <menu>
| SYST <entier>
| KEY ANY
| KEY FM
| FUNCTION ANY;

7.6.5. <menu> ::= <expression de symbole> <option light>
| <expression de symbole > AT <expression de couple> <option light> ;

7.6.6. <caractère> ::= <chaîne>;/* en fait un caractère EBCDIC quelconque */

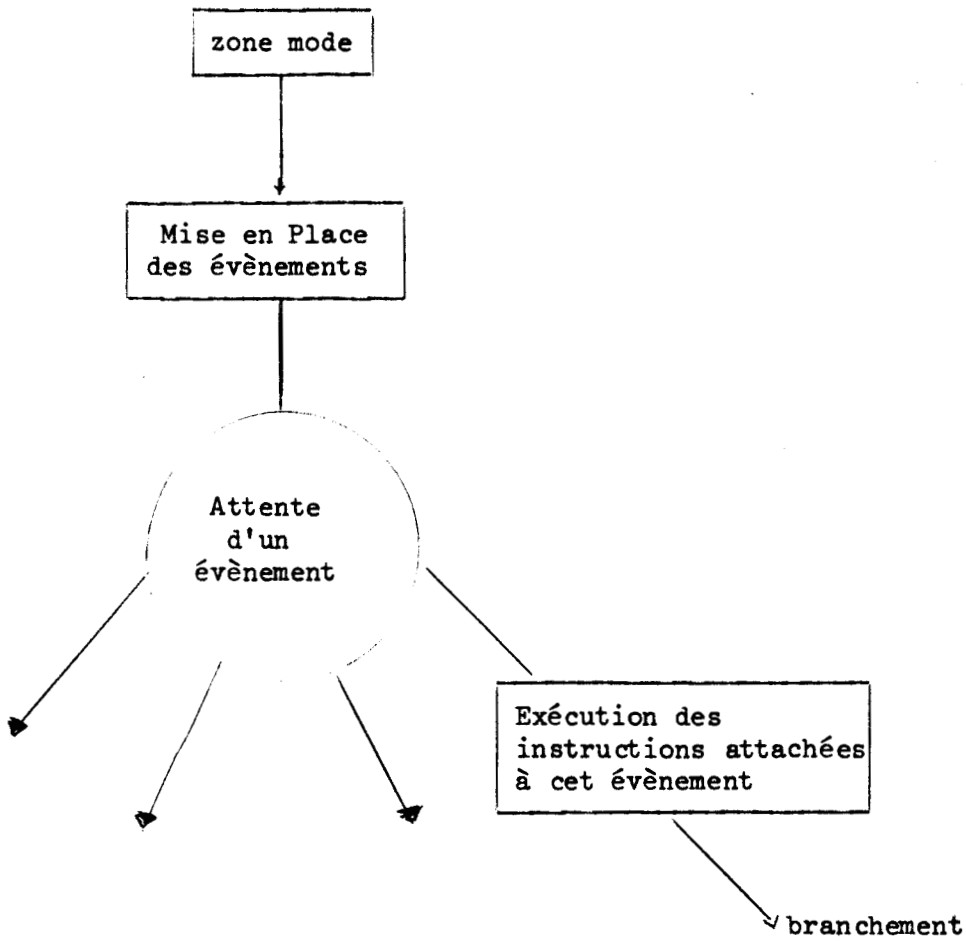
Exemple :

```
  A:=TEXT'*'; COPY A AT <0,0>;
STATE S1;
        I:=0;
TITLE TEXT 'CHOISIR LE MENU';
ON TEXT 'JAUNE' LIGHT 0,0,3 :
        LIGHT A,0,1,3, /* CLIGNOTANT JAUNE  */
        IDEM;
```

```
ON TEXT 'EFFACER' LIGHT 1,0,3:
        DELETE A; GOTO S1;
ON PEN A: I:=I+1; /* Désigner une occurrence de A */
        COPY A AT <I*10,I*10> Light I,0,MOD(I,4);
        IDEM;
```

Sémantique : Un état est une unité de programme accessible uniquement pour un ordre ENTER.

Il peut se décomposer de la façon suivante :



Evènements

Les évènements sont les interventions possibles de l'utilisateur. Nous allons les décrire en examinant successivement tous les moyens de dialogue de la console :

- Clavier alphanumérique : évènements KEY.
  KEY ANY: toutes les touches du clavier sont admises.
  KEY 'caractère'

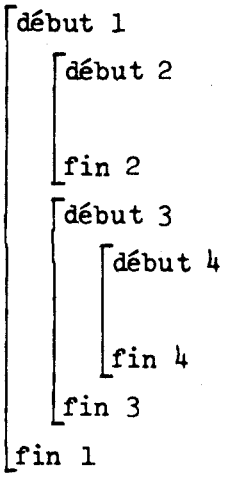
Exemple : KEY'A'. Seule la touche A sera reconnue. KEY FM touche retour chariot, joue un rôle particulier pour le mode EDIT.

- Clavier de fonctions : évènements FUNCTION.
  FUNCTION <entier>. L'entier est compris entre 0 et 31. Seule la touche fonction spécifiée provoquera la reconnaissance de l'évènement.
  FUNCTION ANY : toutes les touches fonction sont admises.

- désignation.
  menus : L'utilisation des menus est analogue à celle des touches du clavier alphanumérique ou de fonctions. Il s'agit généralement de messages indiquant la réponse du système à l'intervention. Les menus sont locaux à l'état et disparaissent de l'écran lorsque le programme change d'état (analogue aux titres). Comme pour les titres, le programmeur peut soit effectuer lui-même la mise en place sur l'écran des menus, soit laisser au système le soin de le faire

  désignation d'un symbole ou d'une image.
  PEN <variable>
  La variable peut être du type SYMBOL, SYMBOL LIST FRAME ou FRAME LIST.
  Lorsqu'une occurence d'un symbole spécifié dans un évènement est désigné, elle devient le HITSYMBOC et le symbole devient le HITSYMBOL. Ceci reste vrai tant que le symbole n'a pas été détruit ou réaffecté ou tant qu'on n'a pas désigné un autre symbole.
  Lorsqu'une image est désignée, la plus petite partie de l'image contenant l'élément vu devient le HITFRAME:

L'image possède une structure de bloc :



Le HITFRAME est la partie d'image comprise entre début i et fin i contenant l'élément vu par le crayon.

Evènements systèmes : SYST n ou n est un entier.

Sont actuellement prévus

- SYST 1 fin de cycle, évènement cyclique se produisant à la fin de chaque parcours de la liste d'affichage par le processeur de la console (1/25 seconde entre deux cycles).

- SYST 2 validation tracking. Cet évènement se produit lors de l'enfoncement du crayon du mode TRACKING (le tracking s'effectue crayon levé). Il est indispensable dans le cas où sont spécifiés dans le même état le mode TRACKING et l'évènement SYST 1 (cycle) car dans ce cas, il est difficile de prendre en compte un autre évènement (à cause de la fréquence du cycle).

Zone mode : Dans cette zone, il est interdit de changer d'état (que ce soit par un branchement ou par un appel de procédure). C'est la seule zone où soient autorisées les instructions de mode.

Distinguons instruction TRACKING ou EDIT et mode TRACKING ou EDIT; l'instruction effectue la mise en place du mode c'est-à-dire que l'on indique au traitement d'interruptions qu'il y aura un mode et les paramètres de ce mode (symbole à trainer, rubberband, par exemple), mais l'exécution du mode ne se fera que lorsque le programme sera en attente d'un évènement.

Incompatibilités entre modes et événements :
- mode TRACKING : tous les évènements désignation sont interdits, le crayon ne pouvant être à la fois en mode poursuite et en mode désignation,
- mode EDIT: tous les évènements KEY sont interdits à l'exception de l'évènement KEY FM qui marque normalement la fin de l'édition de la ligne. Ceci est nécessaire pour empêcher toute ambiguité dans l'édition.

Signification de l'instruction IDEM :

L'instruction IDEM permet de se remettre en attente d'un évènement du même état sans réexécuter la zone mode et la mise en place des évènements Les modes qui étaient suspendus pendant l'exécution des instructions de la zone évènement sont repris. Il est donc nécessaire de ne pas avoir changé d'état (par un appel de procédure) pour exécuter un IDEM.

Exemples :

```
/* SUIVI DU TRACE DEFINI PAR LE DEPLACEMENT DE LA CROIX  */
SYMBOL S; PAIR DEPART;
DEPART:= <0,0>;  S:=NULL;
STATE S1;
TITLE 'DEPLACER LA CROIX';
TITLE 'ENFONCER LE CRAYON SI TERMINE';
TRACKING AT DEPART;
```

```
ON SYST 1: /* A CHAQUE CYCLE MODIFIER LE SYMBOLE */
   S:=S LINE TO CROSSCOOR;
   COPY S AT <0,0>;
   IDEM; /* REPREND LE TRACKING OU IL Y A ETE INTERROMPU */

ON SYST 2:
   EXIT; /* FIN DU TRAITEMENT */

ENDSTATE
   /* EDITION D'UNE CHAINE AVEC POSSIBILITE DE REPRENDRE AU DEBUT  */

   STATE S1
      EDIT 'CHAINE INITIALE';
      TITLE 'FONCTION 0 POUR RECOMMENCER';
   ON KEY FM:
            EXIT; /* ON A FINI  */
   ON FUNCTION 0:
            ENTER S1; /* ON RECOMMENCE TOUT  */
   ON FUNCTION 1:
            IDEM;  /* PAS D'EFFET  */
   ENDSTATE
```

VII.7. STRUCTURE D'UN PROGRAMME

Syntaxe

VII.7.1. <programme> ::= BEGIN <suite de déclarations>';'
<suite d'instructions> ';' <suite d'états> END;

VII.7.2. <suite d'états> ::= <état>
| <suite d'états>';'<état>
| Λ ;

VII.7.3. <corps de procédure> ::= <programme>;

VII.8. FONCTIONS ET PROCEDURES STANDARDS.

Un certain nombre de fonctions et de procédures sont fournies par le système et n'ont pas à être redéclarées.

a/ Fonctions liées aux modes ou aux évènements.

- PAIR PROCEDURE CROSSCOOR;
  /* FOURNIT LES COORDONNEES ACTUELLES DE LA CROIX DE TRACKING */
- PAIR PROCEDURE HITCOOR;
  /* FOURNIT LES COORDONNEES DE L'ELEMENT VU (EN MODE DESIGNATION)*/
- PAIR PROCEDURE HITSYMBCOOR;
  /* FOURNIT LES COORDONNEES DE L'OCCURENCE DESIGNEE (POSITIONNEMENT DU HITSYMBOC) */
- INTEGER PROCEDURE FUNCNAME;
  /* NUMERO DE LA DERNIERE TOUCHE FONCTION ENFONCEE */
- STRING PROCEDURE KEYNAME;
  /* CARACTERE FOURNI PAR LA DERNIERE TOUCHE ENFONCEE DU CLAVIER ALPHANUMERIQUE */
- STRING PROCEDURE INTEXT;
  /* FOURNIT LA DERNIERE CHAINE ENTREE EN MODE EDIT C'EST L'ENFONCEMENT DE LA CLE FM QUI MODIFIE CE TEXTE */
- ITEM LIST PROCEDURE HITNAME;
  /* FOURNIT LA LISTE DES ASSOCIATEURS DONNANT LA HIERARCHIE DE LA DERNIERE IMAGE DESIGNEE. LA QUEUE DE LA LISTE EST L'ASSOCIATEUR LIE AU HITFRAME */
- PROCEDURE UPFRAME;
  /* monte d'un niveau dans la hiérarchie d'une image et modifie le Hitframe, retire donc un élément à la liste fournie par HITNAME */
- INTEGER PROCEDURE HITSYMBCOUNT;
  /* FOURNIT L'INDICE DANS LA LISTE DU SYMBOLE DESIGNE DANS LE CAS OU l'EVENEMENT ATTENDU EST LA DESIGNATION D'UNE OCCURENCE D'UN ELEMENT D'UNE LISTE DE SYMBOLES */
- INTEGER PROCEDURE HITFRACOUNT;
  /* ANALOGUE A LA PRECEDENTE POUR UNE LISTE D'IMAGES */

b/ Fonctions de conversion.

- STRING PROCEDURE CHAR(A);
  INTEGER A (ou REAL A ou ITEM A);
  /* CONVERSION D'UN ENTIER (D'UN REEL OU D'UN ITEM) EN UNE CHAINE DE CARACTERES) */
- INTEGER PROCEDURE INT(A);
  STRING A (ou REAL A);
  /* FOURNIT L'ENTIER DONT LA REPRESENTATION EST LA CHAINE A (ou L'ENTIER IMMEDIATEMENT INFERIEUR AU REEL A) */
- REAL PROCEDURE FLOAT(A);
  INTEGER A; (ou STRING A)
  /* FOURNIT LE REEL DONT LA REPRESENTATION ENTIERE (OU SOUS FORME DE CHAINE) EST A */

c/ Fonctions sur les chaînes de caractères.

- INTEGER PROCEDURE COUNT (A);
  STRING A;
  /* NOMBRE DE CARACTERES DE LA CHAINE */
- STRING PROCEDURE SUBSTR (A,B,C);
  STRING A; INTEGER B,C;
  /* SOUS CHAINE DE A DE LONGUEUR C COMMENCANT PAR LE CARACTERE DE RANG B */
- INTEGER PROCEDURE INDEX (A,B);
  STRING A,B;
  /* FOURNIT L'INDICE DANS LA CHAINE DE LA PREMIERE SOUS CHAINE B, -1 SINON */

d/ Fonctions sur les listes.

- INTEGER PROCEDURE COUNT(L);
  LIST L;
  /* NOMBRE D'ELEMENTS DE LA LISTE L */
- LIST PROCEDURE SUBLST (L,I,J);
  LIST L, INTEGER I,J;
  /* SOUS LISTE DE L DE LONGUEUR J COMMENCANT PAR L'ELEMENT L[I]*/

e/ Fonctions d'entrée-sortie sur télétype.

```
- STRING PROCEDURE INSTR;
    /* CHAINE ENTREE AU TELETYPE */
- PROCEDURE OUTSTR (S);
        STRING S;
    /* IMPRIME LA CHAINE S */
```

f/ Fonctions spéciales.

```
- PAIR PROCEDURE DPMENT (S);
        SYMBOL S;
    /* FOURNIT LE DEPLACEMENT INTRODUIT PAR LE SYMBOLE S */
- REAL PROCEDURE DISTANCE (A,B);
        PAIR A,B;
    /* DISTANCE ENTRE LES POINTS DE COORDONNEES A ET B */
```

g/ Fonctions arithmétiques.

```
- INTEGER PROCEDURE MOD (A,B);
        INTEGER A,B
        RETURN A-(A/B)*B;
- REAL PROCEDURE SQRT (A);
        REAL A; /* √A */
- REAL PROCEDURE SIN(A);
        REAL A;
- REAL PROCEDURE COS(A);
        REAL A;
- REAL PROCEDURE ATAN(A);
        REAL A; /* ARC TG A */
- REAL PROCEDURE LOG(A);
        REAL A; /* LOGARITHME NEPERIEN DE A */
- REAL PROCEDURE EXP(A);
        REAL A; /* e^A */
```

VIII. GESTION DES VARIABLES.

VIII.1. GENERALITES.

Nous distinguons en fait au niveau implantation deux types de variables :

- variables en zone statique dont la place est allouée dès la compilation,
- variables en zone dynamique dont l'allocation est faite dynamiquement au cours de l'exécution.

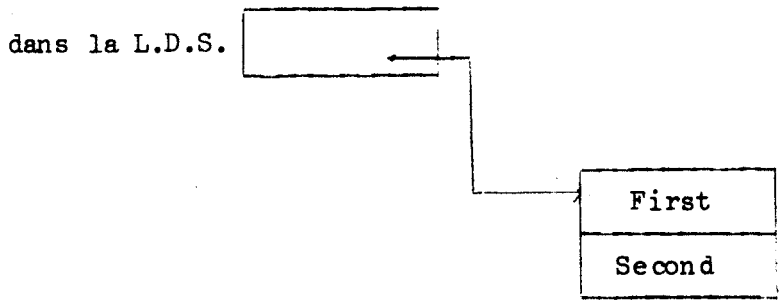
Les variables en zone statique sont les variables de type entier, réel ou item. Les variables de type couple, chaîne, symbôle ou image sont gérés en zone dynamique.

Les variables en zone dynamique sont (à l'exception des couples) des éléments dont on ne peut à priori déterminer la taille. Les couples ont été implantés d'une manière analogue pour rendre homogènes les fonctions d'accès (en particulier, lors de la passation de paramètres de procédures). De plus, ce mode d'accès (par indirection et indexation) est le seul possible sur le MITRA 15 (toute indexation étant nécessairement précédée d'une indirection). Il est par contre pour les entiers, les réels ou les items préférable d'accéder directement à l'élément car il n'y a pas nécessité d'indexation.

VIII.2. GESTION DES COUPLES.

Un couple est implanté de la façon suivante :

En zone statique (Local Data Section du programme ou du sous-programme) un pointeur vers une zone allouée (dans la zone dynamique) de deux mots.



L'allocation est faite lors de la déclaration du couple et la libération ne sera faite qu'en fin de programme ou de sous-programme. La valeur initiale d'un couple est indéfinie.

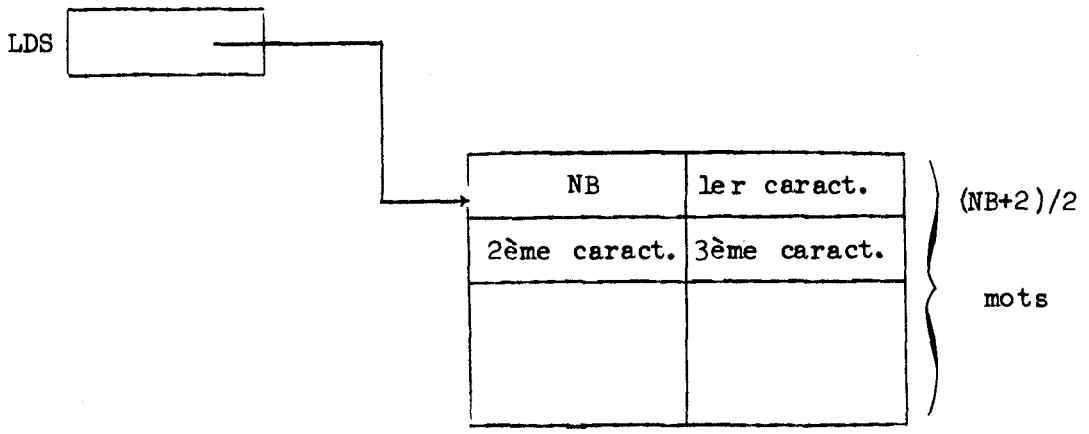
Les modifications de valeur sont effectuées en zone dynamique et le pointeur n'est jamais modifié au cours de l'exécution du programme ou du sous-programme contenant la déclaration.

VIII.3. GESTION DES CHAINES DE CARACTERES.

Les chaînes sont implantées de la façon suivante :

En zone statique, un pointeur vers la chaîne sous la forme suivante : le premier octet est la longueur de la chaîne; les octets suivants contiennent les caractères en EBCDIC.

Le nombre de mots alloués est toujours entier c'est-à-dire qu'il vaut (NB+2)/2 mots pour une chaîne de NB caractères.



Cas de la chaîne vide: le pointeur est nul, pas de zone allouée. La valeur initiale d'une chaîne est la chaîne vide.

Lors de la modification d'une chaîne, on effectue les opérations suivantes :

Calcul de la nouvelle chaîne

|

Libération de l'ancienne chaîne

|

Allocation de la nouvelle chaîne (NB+2)/2 mots

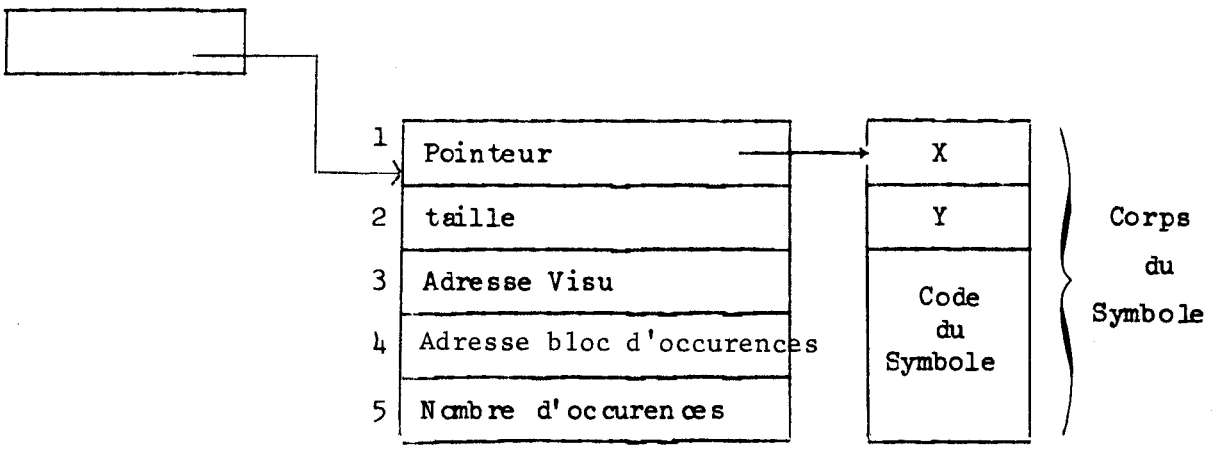
|

Transfert de la chaîne depuis le buffer vers la zone allouée

En fin de programme ou de sous-programme les chaînes déclarées dans ce programme sont libérées.

VIII.4. GESTION DES SYMBOLES:

Le pointeur situé en zone statique pointe vers un bloc alloué de 5 mots appelé descripteur du symbole et contenant les informations suivantes :



Le 1er mot du descripteur pointe vers le corps du symbole.

Le 2ème mot contient la taille en mots du corps du symbole.

Les 3 mots suivant contiennent l'adresse relative à la mémoire du processeur du symbole, de son bloc d'occurences et le nombre d'occurences du bloc.

L'utilisation de ces 3 mots sera détaillée dans le paragraphe suivant.

Le corps du symbôle est formé :

a/ de 2 mots indiquant le déplacement du faisceau introduit par le symbôle. L'utilisateur peut connaître cette information en utilisant la fonction DPMENT.

Lors de la déclaration, le descripteur du symbôle est alloué et son contenu est mis à zéro.

Modification d'un symbôle :

- libérer (éventuellement) la mémoire visu (voir § IX): opération analogue à DELETE.

- Calcul du nouveau symbôle : Le code visu est exprimé en déplacements relatifs. Le déplacement du faisceau et la taille du symbôle sont évalués au fur et à mesure de la construction.

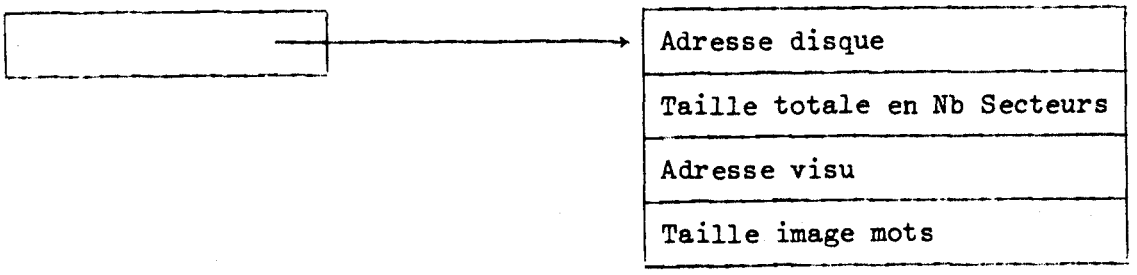
- libération de l'ancien symbôle.

- allocation et transfert du nouveau symbôle.

En fin de procédure : on libère la mémoire visu, le symbole et le descripteur.

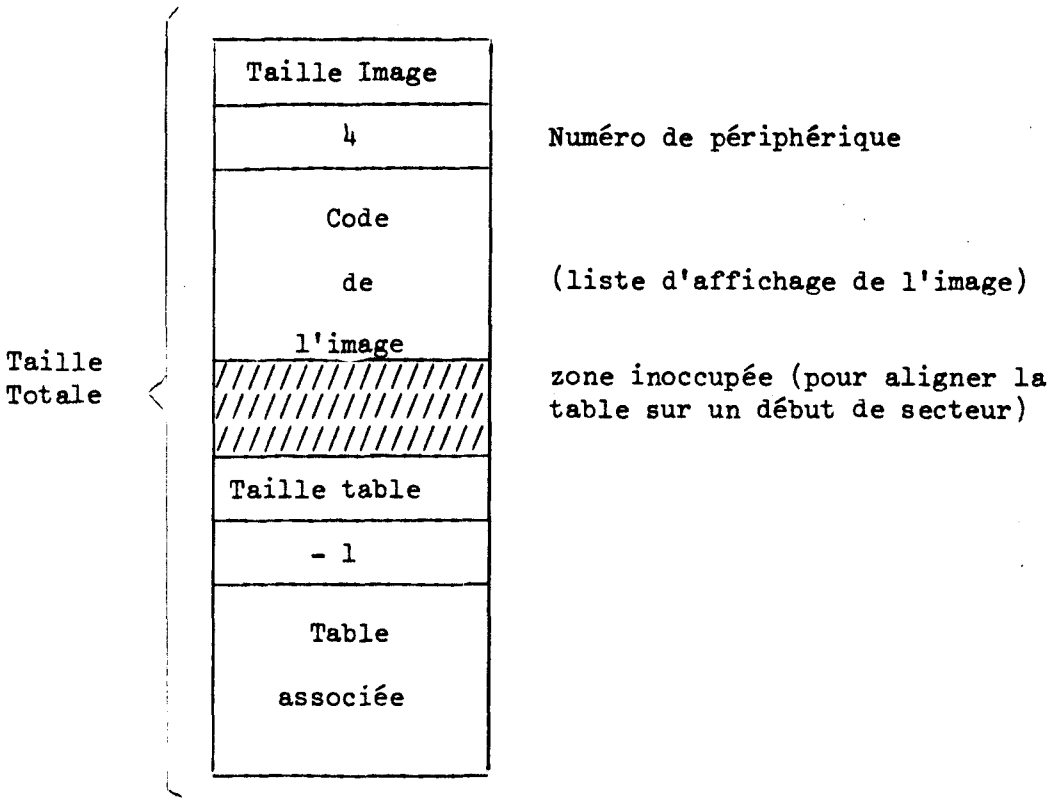
VIII.5. GESTION DES IMAGES.

Le pointeur en zone statique pointe sur un bloc alloué de 4 mots appelé descripteur de l'image.



Les images sont générées par les procédures externes et stockées sur le disque du Mitra.

Structure d'une image sur disque



Lors d'une déclaration, il faut allouer le descripteur et le mettre à zéro.

A la fin de la procédure il faut libérer la place en mémoire visu, l'espace disque et le descripteur.

Les modifications du descripteur seront décrites dans les paragraphes IX et XI, les images n'étant modifiées que par les procédures externes.

VIII.6. GESTION DES TABLEAUX.

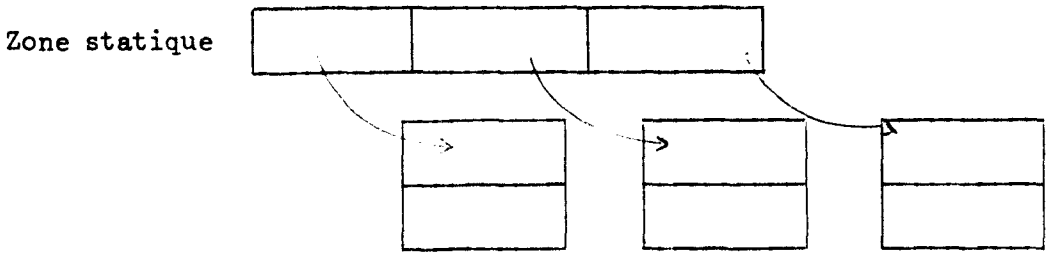
Les tableaux sont gérés en zone statique, c'est-à-dire qu'un emplacement mémoire est réservé pour chaque élément dans cette zone. Si le tableau est composé d'entiers de réels ou d'items, la valeur est stockée directement dans cet emplacement, sinon on trouve un pointeur vers la zone dynamique, les éléments respectant alors la structure ci-dessus.

Les tableaux ont été introduits à la demande des utilisateurs qui sont habitués à les employer dans d'autres langages de programmation (FØRTRAN, ALGØL ou PL1). Le caractère statique de ce type de structure en limite l'utilisation et est insuffisant dans la majorité des cas. C'est pourquoi nous estimons que les listes sont indispensables. Cependant, il est parfois plus agréable, en particulier, lorsque la structure est figée, d'utiliser des tableaux.

Exemples :

Tableau de 3 entiers

Tableau de 3 couples



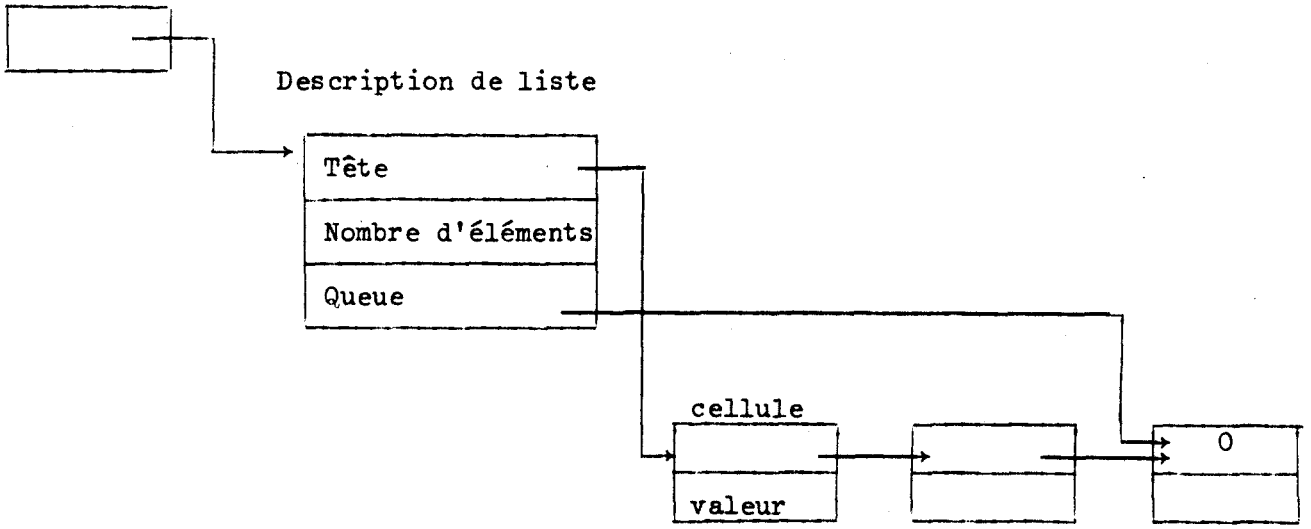
VIII.7. GESTION DES LISTES.

Dans la plupart des applications interactives, il est indispensable de pouvoir créer et manipuler des structures de données offrant une grande souplesse. En particulier, le nombre d'éléments de la structure et la répartition des éléments à l'intérieur de cette structure ne doivent pas être figés.

La structure doit cependant, rester relativement simple, le Mitra 15 n'ayant pas la puissance suffisante pour gérer des structures complexes. Ces considérations nous ont amenés à choisir la liste comme mode de structuration dynamique des données. Il faut noter qu'il s'agit d'un mode relativement simple (les listes sont linéaires et homogènes) et qui ne permet pas en général d'implanter la structure de données nécessaire à l'application dans le calculateur satellite.

Les listes sont implantées de la façon suivante :

En zone statique



Si la liste est vide, le descripteur contient 3 mots nuls. Les éléments de la liste sont formés de cellules de deux mots, le premier étant un pointeur vers l'élément suivant, le second (valeur) étant soit directement la valeur (entier, réel ou item) soit un pointeur vers la valeur de l'élément (couple ou chaîne), soit un pointeur vers un descripteur (image ou symbôle).

Lors de la déclaration d'une liste on alloue le descripteur. En fin de procédure, on libère les éléments, les cellules et le descripteur.

IX. GESTION DE LA MEMOIRE D'ENTRETIEN ET ACTIONS SUR LES ELEMENTS DE LA LISTE D'AFFICHAGE.

IX.1. GESTION DE LA MEMOIRE D'ENTRETIEN.

Contraintes :

- La structure de la liste d'affichage doit tenir compte du tri-couleur c'est-à-dire que tout changement de couleur doit être suivi d'un positionnement absolu et que les sous-programmes ne contiennent ni changements de couleur, ni positionnements.

- La reconnaissance de l'élément désigné doit être la plus rapide possible. Cette reconnaissance est facilitée par l'utilisation de la pile Hardware.

- Les opérations sur la mémoire doivent être transparentes à l'utilisateur celui-ci étant intéressé uniquement par le résultat visuel de l'opération.

- Eviter les erreurs de l'utilisateur en n'autorisant que les moyens d'entrée effectivement spécifiés dans l'état et en interdisant tous les autres, ceci pour éviter des interruptions parasites.

Structure de la méloire d'entretien :

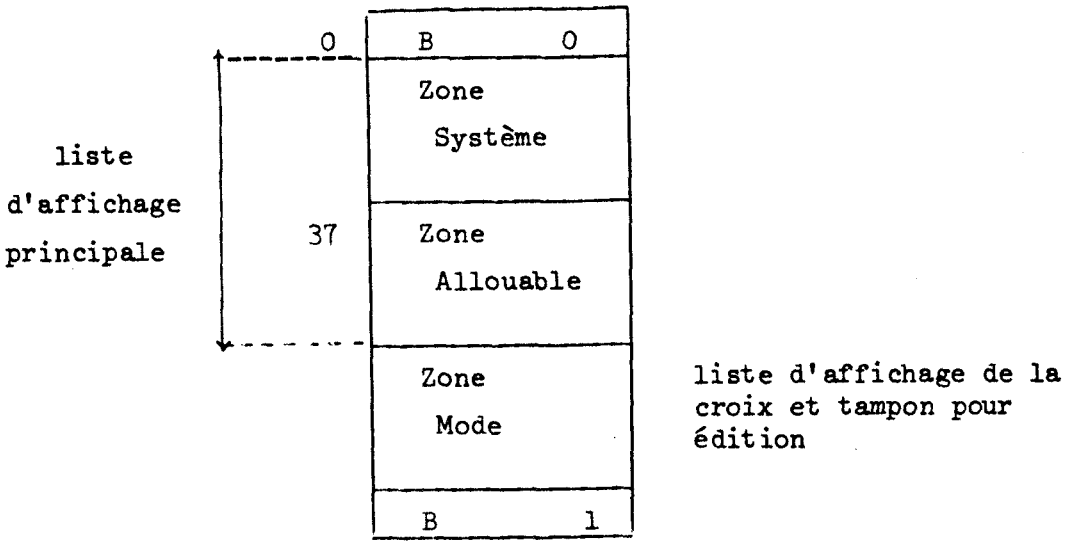


Fig. IX.1

Abréviations utilisées

M V	: mot vide
B	: branchement
IME	: inhibition moyen d'entrée
RME	: reconnaître moyen d'entrée
CHC	: changement de couleur
EMP	: empiler
DEP	: dépiler
POS X	: positionnement X
POS Y	: positionnement Y
BSP	: branchement vers sous-programme.

IX.2. STRUCTURE DE LA LISTE D'AFFICHAGE PRINCIPALE.

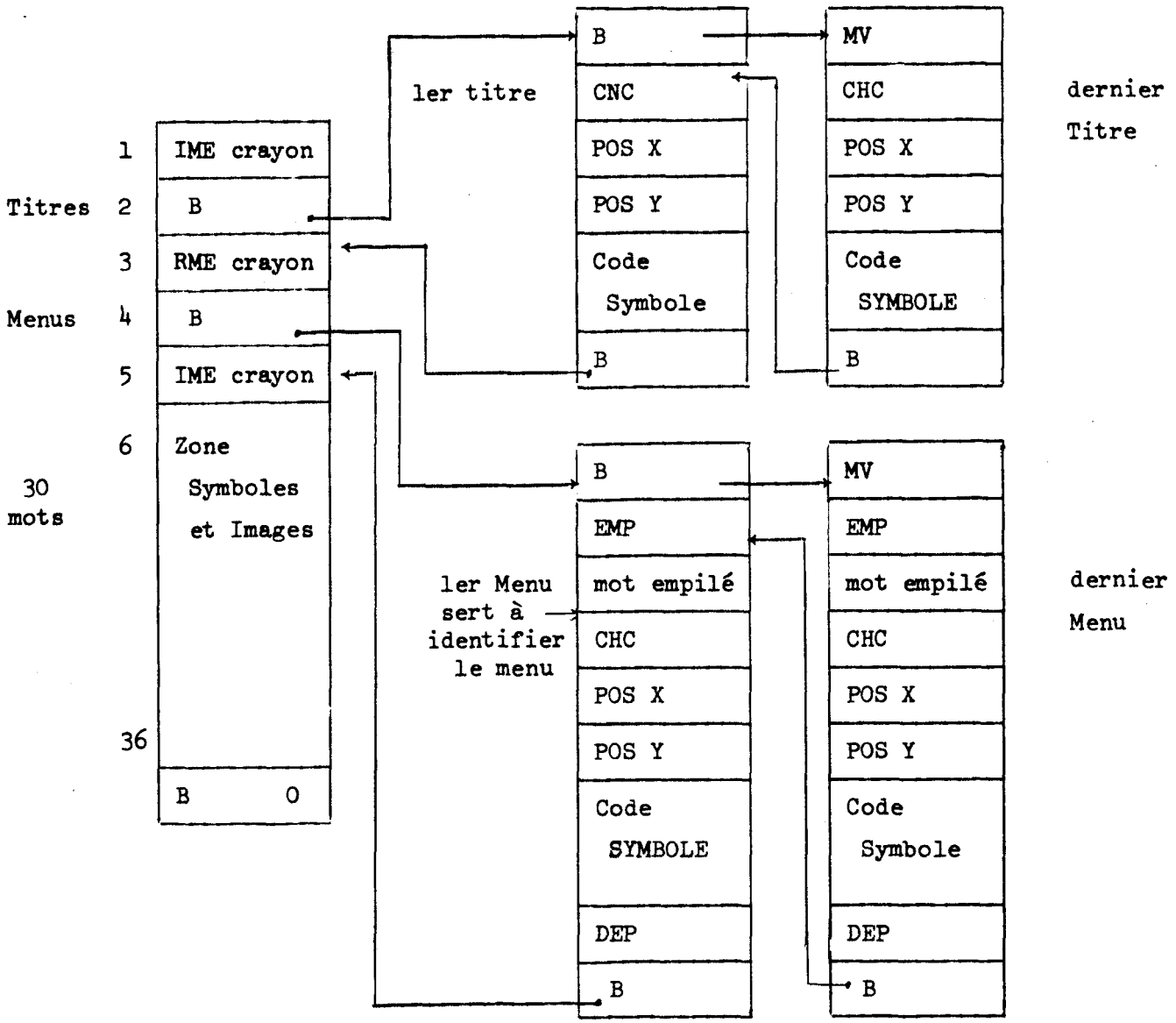


Fig. IX.2

Liste d'affichage principale : partie titres et menus. S'il n'y a pas de titres ou de menus le branchement correspondant est remplacé par un mot vide. Le schéma ci-dessus se généralise aisément au cas de n titres ou n menus.

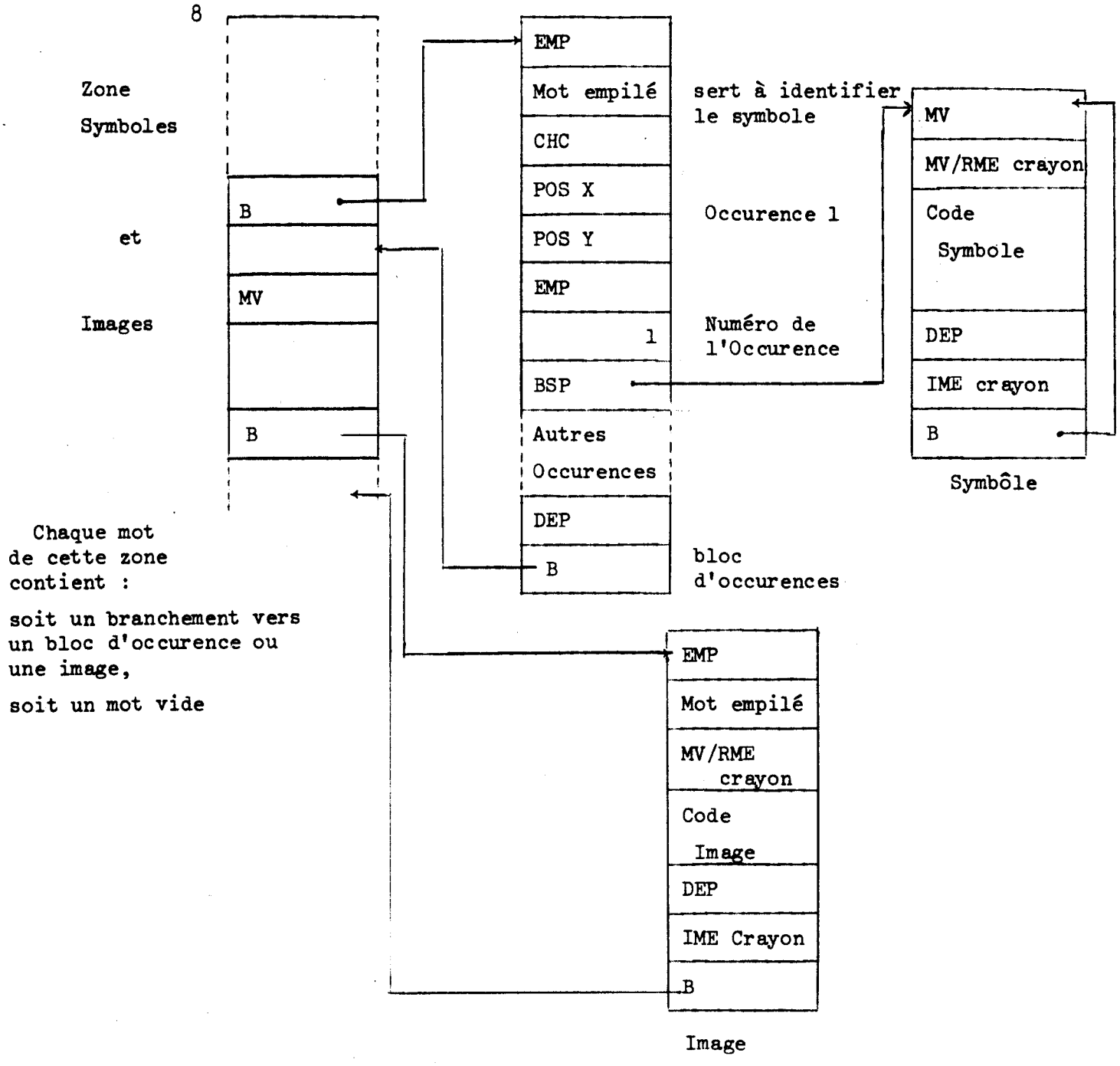


Fig. IX.3. Liste d'affichage principale
Partie symbôles et images

Si une occurence est libre le BSP est remplacé par un ordre DEP.

IX. GESTION DES TITRES ET DES MENUS.

Titres et menus sont locaux à un état, c'est-à-dire que lors d'un changement d'état, il faut les libérer.

- les titres ne sont pas désignables (IME crayon précédant le mot Titres),
- les menus sont toujours désignables (RME crayon précédant le mot Menus).

Le code d'un titre ou d'un menu est analogue comme structure à celui d'un symbôle (voir Paragraphe VII) c'est-à-dire qu'il ne comporte pas de changements de couleur, ni de positionnements ou de tracés en absolus.

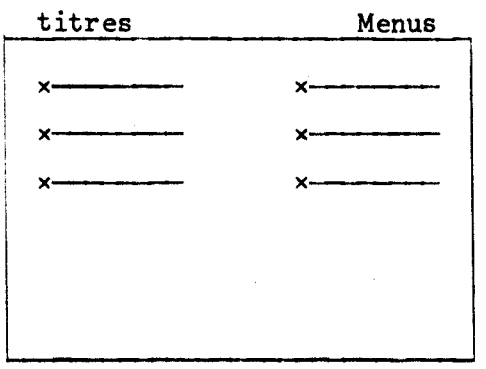
Remarque sur les options par défaut.

L'utilisateur peut préciser le positionnement et les caractéristiques de ses titres ou de ses menus ou laisser au système le soin de s'en charger. Les caractéristiques par défaut des titres et menus sont :

- couleur verte,
- intensité normale,
- pas de clignotement.

Positionnement :

Titres	1er	<-500,500>
	suivants	+< 0,-10>
Menus	1er	< 250,500>
	suivants	+<0,-10>



IX.4. GESTION DES SYMBOLES.

Un symbôle est un sous-programme graphique qui se trouve en mémoire de la visu uniquement lorsqu'il est utilisé, c'est-à-dire lorsqu'il possède des occurences. Son code ne contient aucun changement de couleur, aucun positionnement ou tracé de vecteurs absolus. Un bloc d'occurences ne contient que des occurences d'un seul symbôle.

Action des instructions sur les symbôles :

- COPY : créé une nouvelle occurence du symbôle.

Il y a plusieurs cas possibles :

a/ pas encore d'occurences.
Il faut créer un bloc d'occurences et transférer le bloc d'occurences et le symbôle.

b/ le symbôle a des occurences.
Il faut lire le bloc d'occurences pour chercher une occurence libre (DEP au lieu de BSP dans le bloc). Si on en trouve une mettre à jour le changement de couleur, le positionnement et le branchement. Sinon il faut agrandir le bloc d'occurences (le nombre maximum d'occurences d'un symbôle est 32).

- MOVE et LIGHT agissent uniquement sur les positionnements et changements de couleur se trouvant dans le bloc d'occurences.

- DELETE : action sur un symbôle: libération du symbôle et de son bloc d'occurences.

action sur l'occurence désignée: on remplace le BSP correspondant par l'ordre DEP.

- lors d'une réaffectation de symbôle ou en fin de procédure (dans laquelle le symbôle est déclaré), le symbôle et le bloc d'occurences sont libérés.

IX.5. GESTION DES IMAGES.

Les images sont générées par les procédures externes. Leur code est en absolu et contient des changements de couleur et des positionnements. Elles ne peuvent donc pas être appelées comme sous-programmes graphiques.

Remarque : Tout changement de couleur est suivi d'un positionnement pour éviter tout ennui avec le tri-couleur

Action des instructions sur les images :

- DISPLAY : transfert de l'image depuis le disque vers la mémoire visu.
  2 cas possibles :
  - l'image était déjà en mémoire visu, il n'est pas nécessaire de demander une allocation, on réutilise la place occupée (utile pour restaurer une image dans son état initial).
  - l'image n'est pas en mémoire visu, il faut allouer la place.

- DELETE : libération de l'image (→ extinction).

- Light : c'est une opération longue; il faut modifier tous les positionnements et tous les changements de couleur contenus dans l'image.

Cas du Hitframe

- DISPLAY : transfert de la partie d'image correspondante depuis le disque vers la mémoire visu (remplacement de l'ancien code).

- DELETE : on remplace la première instruction du Hitframe par un branchement vers le mot suivant la fin du Hitframe.

- Light : même opération que pour une image complète mais uniquement sur la partie d'image désignée.

Les images sont libérées lors d'une réaffectation (au retour d'une procèdure externe) ou à la fin de la procédure où elles ont été déclarées.

X. GESTION DES EVENEMENTS ET DES MODES.

X.1. INTRODUCTION.

Nous distinguons en fait 2 opérations :

1/ Mise en place des modes et des évènements,

2/ attente et reconnaissance d'un évènement.

Pour faciliter la reconnaissance des évènements crayon et ne pas effectuer de lecture en mémoire visu pour effectuer cette reconnaissance nous utiliserons la pile cablée.

Fonctionnement de la pile :

durant le parcours de la liste d'affichage, l'information contenue dans le mot suivant l'ordre EMP est stockée dans la pile et est retirée à la rencontre du mot DEP. Lors d'une interruption crayon, le parcours de la liste d'affichage est arrêtée et le système reçoit le contenu de la pile.

Pour les autres évènements est communiquée la nature de l'interruption (exemple : cycle, fin de, clavier ...) et l'identificateur du moyen de dialogue (caractère dans le cas d'un évènement clairci, numéro de la touche fonction).

Si l'évènement a été spécifié dans la description de l'état, il doit être reconnu et la séquence attachée doit être exécutée. Pour limiter les interruptions parasites (par exemple : fonction enfoncée alors qu'il n'existe pas d'évènement fonction, désignation d'un élément non prévu dans l'état ...), les types d'évènements non prévus sont inhibés.

X.2. MISE EN PLACE DES ETATS.

La mise en place d'un état comporte :

a/ mise en place des modes

- création des titres,
- initialisation du tracking, modification de la position de la croix. Attacher une occurence d'un symbôle à la croix. Initialisation du rubberband.
- initialisation de EDIT, positionnement du texte, envoi du texte initial.

b/ mise en place des évènements.

La table des évènements a une longueur de 25 mots. Cette longueur est largement suffisante car le débordement signifierait un choix pour l'utilisateur parmi 26 interventions possibles, ce qui est trop important.

Informations rangées dans la table des évènements :

1 évènement = 1 mot		ler octet	type de l'évènement,
		2ème octet	permet l'identification de l'évènement
type :	signification		information contenue dans le 2ème octet
# FF	SYST		n° de l'évènement système
# F1	FUNCTION		n° de la touche ou # FF si ANY
# F0	KEY		code EBCDIC du caractère ou # FF si ANY
# 00	menu		index dans la table des évènements
# 04	Symbôle		Indice dans la table crayon
# 08	Image		
# 06	Liste de symbôles		
# 0A	Liste d'images		

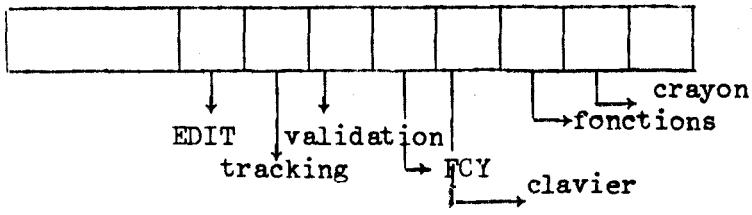
table crayon

Cette table contient des pointeurs vers les descripteurs des éléments correspondants (symboles, images ou listes)

Les informations contenues dans la table des évènements sont pour les évènements crayon rangées dans la liste d'affichage (voir figures IX.2 et IX.3) pour faciliter l'identification.

A la fin de la mise en place, le programme envoie un mot contenant les types d'évènements autorisés au traitement d'interruption et les modes à prendre en compte et se met en attente

contenu du mot



Le traitement des interruptions se charge alors de l'exécution des modes (interruption Light pen en mode tracking, clavier en mode EDIT) et rend la main au programme d'interactions lorsqu'un évènement appartenant aux types spécifiés se produit.

X.3. RECONNAISSANCE D'UN EVENEMENT.

a/ cas d'un évènement système, clavier ou fonction simple: il suffit de parcourir la table jusqu'à ce qu'on trouve la spécification du même évènement, l'index dans la table donnant l'index dans la table de branchement associée.

b/ cas d'un évènement crayon: le traitement d'interruptions communique
.l'adresse dans la liste d'affichage de l'élément
.les coordonnées de l'élément
.le contenu et la hauteur de la pile Hardware.

Le dernier mot de la pile est toujours un mot de la table.

a/ cas d'un menu : la pile contient un seul mot, il n'est pas nécessaire de consulter la table, ce mot donnant directement l'adresse de branchement.

b/ Cas d'un symbôle : la pile contient 2 mots. Il faut identifier le symbole et l'occurence désignée.

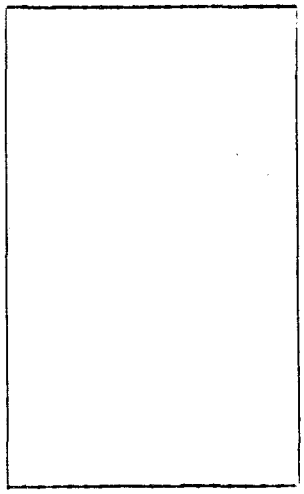
Recnerche de l'index de branchement dans la table par comparaison entre le dernier mot empilé et le contenu de la table.

Recherche du Hitsymbol: le 2ème octet du mot donne l'index dans la table crayon - le mot ainsi repéré donne un pointeur vers le descripteur du symbôle.

Recherche de Hitsymboc :
Hitsymboc est un pointeur vers l'occurence. Le 1er mot empilé donne le numéro de l'occurence.
Hitsymboc : Adresse bloc d'occurence + 2 + 5*(Numéro de l'occurence -1)
HITSYMBCOOR est trouvé en lisant les mots à l'adresse Hitsymboc +1, on range les coordonnées de l'élément vu dans Hitcoor.

c/ Cas d'une image : Il faut identifier l'image et la partie d'image vue (Hitframe)

Structure de l'image et de la table associée :



l'image a une structure de bloc, un début de bloc étant constitué de la façon suivante :

EMP
déplacement depuis le début
CHC
POS X
POS Y

et une fin de bloc par l'ordre DEP.

Le Hitframe est constitué par la plus petite partie d'image contenant l'élément vu, c'est-à-dire que le sommet de la pile contient le déplacement depuis le début de l'image du Hitframe et les mots suivants (sauf le dernier) décrivent la hiérarchie de l'image.

Structure de la table associée (sur disque)

Taille	-1	table proprement dite

La table a la même structure que l'image mais contient uniquement les débuts et fins de bloc de la façon suivante :

début de bloc :	déplacement depuis le début de l'image (mot empilé)
fin de bloc :	- déplacement depuis le début de l'image

Recherche de l'image désignée :

- le dernier mot empilé contient une information dont l'index dans la table des évènements donne l'index dans la table de branchements et dont le 2ème octet donne l'index dans la table crayon d'un pointeur vers le descripteur de l'image.

Il faut alors lire la table associée et créer la liste Hitframe (liste de Hauteur de pile -1 éléments) contenant les items associés aux différentes parties d'image identifiée par la pile. De plus, on crée une table qui contient pour chaque partie d'image l'adresse de début et l'adresse de fin de la partie d'image concernée.

Le Hitframe est le dernier élément de cette table.De plus, on range dans HITCOOR les coordonnées de l'élément vu.

Rôle de la procédure UPFRAME

Active le dernier élément à la liste HITNAME.
Le Hitframe est modifié (on pointe sur l'élément précédent).

XI. EXECUTION DE PROCEDURES EXTERNES.

XI.1. INTRODUCTION.

Il s'agit de lancer l'exécution de procédures sur un calculateur autre que le Mitra 15 en lui passant des paramètres et en récupérant les résultats et les images générées par ces procédures.

Dans le système METAVISU, le procédé utilisé avait été rendu complètement transparent à l'utilisateur grâce aux adaptations suivantes du système :

- L'analyseur du langage d'interaction génère un programme principal écrit en CPL/1 qui appelle les procédures externes. Ce programme contient des ordres spéciaux réservés à l'usager visu qui permettent de lire des données ou d'écrire des résultats dans un buffer (page collée) ou de se mettre en attente d'un ordre provenant du processus premier visu.

- Le processus premier visu se charge de vérifier la présence de l'interpréteur CPL/1 lors d'un appel et en effectue éventuellement le chargement.

- Lorsqu'une procédure externe est en cours d'exécution, l'utilisateur de la console se trouve dans la même situation qu'un usager normal du système (en particulier, la console est considérée comme un télétype). Il peut interrompre l'exécution et entrer des commandes pour l'interpréteur CPL/1.

Cette technique n'était pas réutilisable pour les raisons suivantes :

- la génération d'un programme symbolique n'est pas possible si on veut laisser le choix du langage utilisé pour les procédures externes à l'utilisateur.

- Il n'est pas possible sans SIRIS 7 de lancer un programme dans la partition time-sharing depuis un programme temps réel ou depuis le MITRA 15 (le Time Sharing travaillant uniquement avec des lignes en mode "caractère").

XI.2. DESCRIPTION SOMMAIRE DE LA LIAISON ET DE QUELQUES CARACTERISTIQUES DE SIRIS 7.

La liaison entre les deux calculateurs est constituée d'une ligne téléphonique fonctionnant à 48000 bauds. Elle est gérée suivant la procédure TMM-UC in Half duplex (transfert mode message d'unité centrale à unité centrale), les blocs d'informations échangés ayant une taille maximale de 225 octets (limitation imposée par le SGT 15).

SIRIS 7 est un système de multiprogrammation, la priorité respective entre les différentes tâches étant déterminée par leur classe et l'importance des ressources qu'elles réclament; une tâche ne pouvant être activée que si elle dispose de toutes les resscurces dont elle a besoin. Il n'est pas possible de charger les procédures externes à chaque appel sous peine de pénaliser gravement l'utilisateur sur le plan temps de réponse, le temps d'attente au chargement pouvant être intolérable (supérieur à 10 minutes par exemple).

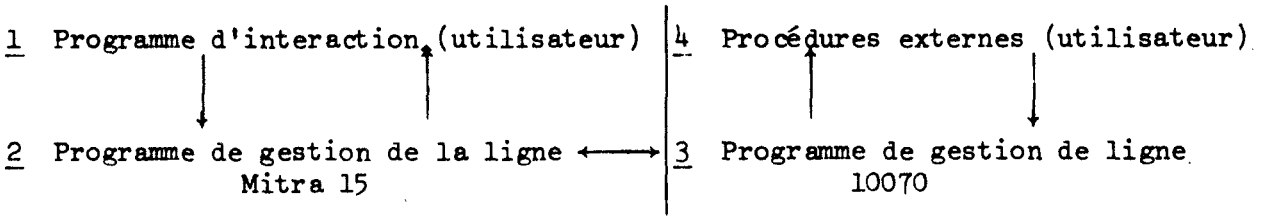
Le programme utilisateur ne doit pas être chargé de gérer la ligne car il peut contenir des erreurs et aucune récupération ne serait possible dans ce cas.

Les primitives utilisées dans les programmes utilisateurs doivent être simples. En particulier, les mécanismes de communication sont gérés par l'utilisateur dans les procédures externes. Il faut donc qu'ils soient clairs et aisés à comprendre.

XI.3. ORGANISATION GENERALE DU MECANISME DE DECLENCHEMENT D'UNE PROCEDURE EXTERNE.

Nous décrivons le mécanisme dans le cas où les procédures externes sont écrites dans un langage pouvant être compilé et pouvant appeler des sous-programmes écrits dans d'autres langages. Ce mécanisme doit être modifié dans le cas de l'utilisation d'un interpréteur. Il y aurait chargement à chaque appel.

Nous avons en présence quatre processus :



Les processus 2 et 3 doivent être complètement transparents à l'utilisateur.

Le mécanisme adopté est indépendant de la manière dont est chargé le programme contenant les procédures externes, l'utilisateur pouvant bien entendu déclencher ce chargement depuis le MITRA 15. Une fois ce programme chargé, il reste actif jusqu'à ce que le programme d'interaction ou l'utilisateur décide qu'il est terminé, ceci pour éviter le rechargement qui serait long (à moins d'utiliser SIRIS 7 en mono-programmation, ce qui serait pire encore).

Tâches des différents processus :

1 .Préparer les données.
.Déclencher le programme de gestion de ligne.
.Attendre la fin d'exécution de ce programme
.Récupérer les résultats et les images.

2 .Transférer les données.
.Permettre à l'utilisateur d'activer des tâches sur le 10070 et de les surveiller.
.Récupérer résultats et images.

3 .Transfert de données ou de résultats.
.Déclencher et surveiller l'exécution des procédures externes.
.Activer des tâches dans le 10070.

4 .Effectuer les calculs et générer les images.

Les processus 2 et 3 sont des programmes temps réels.

Le processus 2 est attaché à un niveau d'interruption et sera déclenché par une activation de cette interruption.

Le processus 3 est un programme de classe F, ce qui lui permet d'une part de ne pas avoir de limite de temps et d'utiliser les procédures privilégiées de SIRIS 7,en particulier l'horloge temps réel.

Nous allons décrire indépendamment chacun des processus puis nous montrerons quel est le cheminement normal lors du déclenchement d'une procédure externe.

XI.4. APPEL DE PROCEDURE EXTERNE DANS LE PROGRAMME D'INTERACTION.

Le mécanisme est simple :

Les données seront transférées sous la forme de chaînes de caractères pour simplifier les interfaces. Il y a donc conversion de chacune des données suivant les formats suivants : I 6 pour les entiers ou les items et E 12.4 pour les réels, les chaînes de caractères n'étant évidemment pas converties. Ces données sont stockées dans le fichier de travail (fichier géré en accès direct et dont les secteurs sont alloués au fur et à mesure des besoins, ce fichier contient également les images). Le programme d'intéraction range dans la zone commune (zone système accessible par tous les processeurs) les informations suivantes : l'ordre, ensuite l'adresse disque et la taille en secteurs des données, puis l'adresse disque et la taille des différentes images susceptibles d'être réaffectées (cette adresse et cette taille étant nulle si l'image n'a pas encore été affectée). Il ferme le fichier de travail.

Il excite ensuite l'interruption à laquelle est connecté le programme de gestion de ligne et se met en attente.

A la fin du programme de gestion de ligne, il est réveillé, retrouve dans la zone commune, l'adresse et la taille des résultats, qu'il affecte aux variables spécifiées par l'utilisateur. S'il n'y a pas concordance entre les résultats envoyés et attendus, il sort un message

mais continue en séquence. Il affecte ensuite les différentes images (modification des informations contenues dans le descripteur de chaque image).

XI.5. PROGRAMME DE GESTION DE LIGNE COTE MITRA 15.

Nous le décrirons sous forme de diagrammes d'état et en pseudo-langage d'interaction. Par souci de clarification nous ne décrirons pas les transferts de données, de résultats ou d'images ni les mécanismes de détection d'erreurs systèmes (panne hardware, programme de gestion de ligne 10070 avorté ou erreur système SIRIS 7), ces erreurs étant généralement irrécupérables.

Description de l'ordre envoyé par le programme d'interaction :

Le programme peut envoyer deux ordres différents : 'F' fin d'exécution des procédures externes, soit 'L' lancement.

les caractères suivants 'L' pouvant être

'D' s'il y a des données blanc sinon

'R' si le programme attend des résultats blanc sinon

'I' si le programme attend des images blanc sinon.

Diagramme d'état :

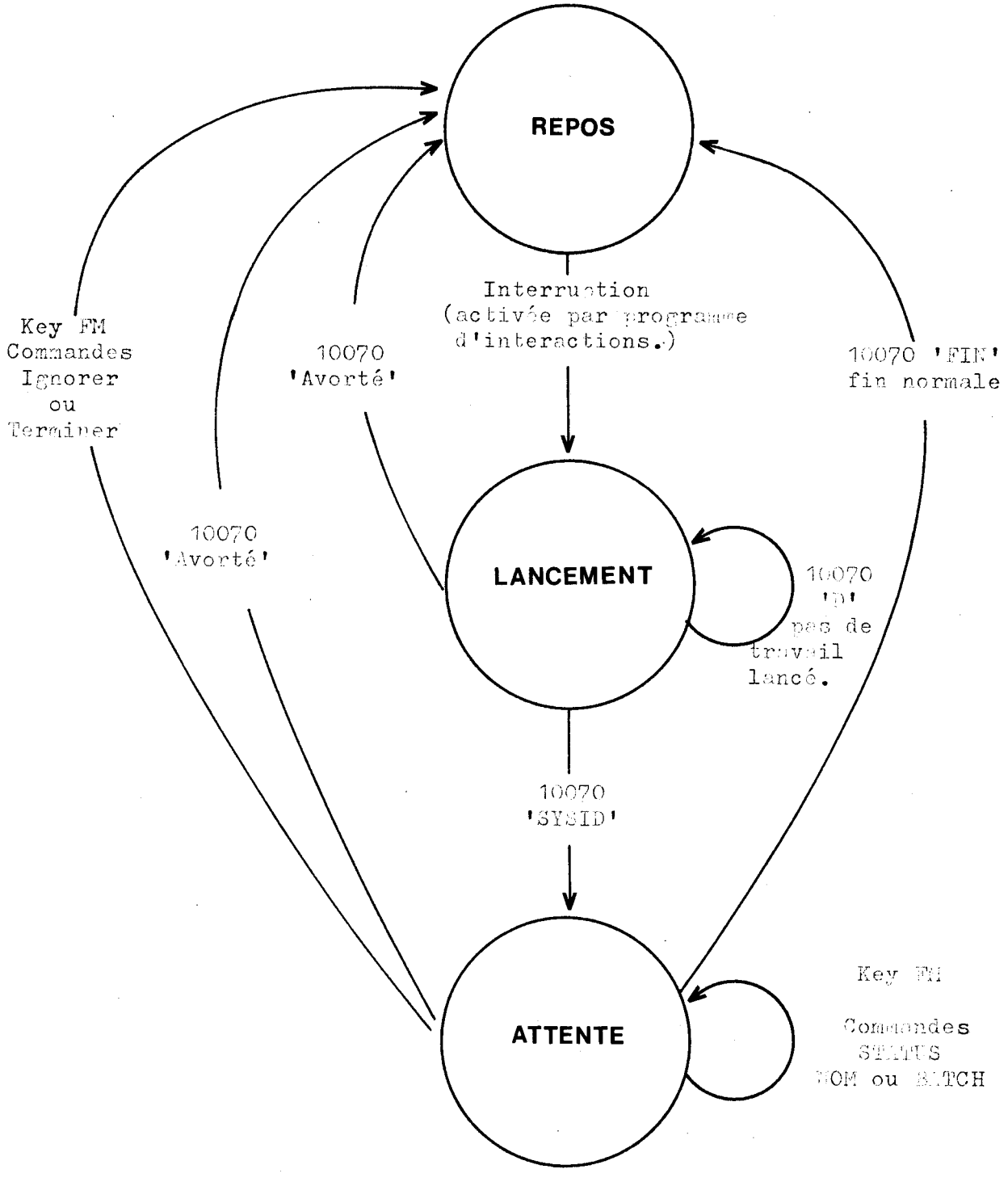


Fig. XI.1

Programme :

```
Begin Integer LIGNE /* identification de la ligne */, TRAV / *identificateur
  du fichier de travail */; Integer I;
  Ref String procedure REPONSE;
  Ref procedure DONNEES; /* transfert des données vers le 10070 */
  Ref procedure RESULTATS ;  /* récupération des résultats depuis le 10070 */
  Ref procedure IMAGES; /* récupération des images depuis le 10070 */
  String A,B; String ORDRE /* ordre envoyé par le programme d'interaction */
  OPEN (LIGNE); /* ouverture de la ligne */
  State REPOS; /* état initial on attend l'interruption pour commencer */
  ON I
    OPEN (TRAV); /* ouvrir le fichier de travail */
    I : = 0;
    WRITE (ORDRE); /* envoi de l'ordre */
    ENTER LANCEMENT;
  endstate

State LANCEMENT;
  READ (A); /* instruction de mode, l'évènement associé étant la fin
            de transmission (10070), analogue à Edit */
  on 10070 :
    if A = 'AVORTE' then Begin /* cas d'avortement du programme utilisation */
                         Outstr (A); /* prévenir l'utilisateur */
                         Close (TRAV); /* fermeture du fichier */
                         enter REPOS; /* terminé, le programme d'interaction
                                      reprend la main */
                         End;
    if A = 'D' then begin /* pas de programme utilisateur */
                    Outstr ('NOM DU PROGRAMME ?'):
                    B : = INSTR; /* réponse de l'utilisateur */
                    I : = 1;
                    Enter RANGEMENT;
                    End;
    outstr (A); /* donner l'identificateur du programme à l'utilisateur */
```

```
If I = 1 then WRITE (ORDRE); /* répéter ordre */
If substr (ordre, 2,1) = 'D' then DONNEES; /* transfert des données */
enter ATTENTE; /* le programme utilisateur est lancé */

Endstate;

State ATTENTE;

TITLE TEXT 'PROGRAMME LANCE' AT  <-500,-500> light 1,0,2;  /* message
      pour l'utilisateur à la console */
edit; /* attente commande secondaire éventuelle;
READ (A);
ON 10070 :
  if A = 'AVORTE' then
              begin /* cas d'avortement */
              Outstr (A);
              CLOSE (TRAV);
              enter REPOS;
              end;
         /* ici tout s'est bien passé */
  if substr (ORDRE, 3,1) = 'R' then RESULTATS; /* récupération de résultats */
  if substr (ORDRE, 4,1) = 'I' then IMAGES; /* récupérer les images */
  CLOSE (TRAV); /* fin normale */
  Enter REPOS;  /* l'utilisateur peut continuer */
on key FM :  /* commande utilisateur */
  WRITE (intext); /* envoi de la commande */
  B : = substr (Intext, 1,1);
  If B = 'I' | B = 'F' Then begin /* commandes    ignorer    ou terminer */
                              CLOSE (TRAV);
                              Enter REPOS;
                              end;
  If B = 'S' | B = 'N' | B' = 'B' then outstr (REPONSE); /* écrire la réponse */
  enter ATTENTE;
endstate;
end /* du programme */
   String Procedure REPONSE;
```

```
  begin string A;
  State REPONS;
  READ (A);
  on 10070: Return A;
endstate;
end /* reponse */;
```

References : SGT 15 manuel d'utilisation; SGF 15 manuel d'utilisation
MTRD manuel d'utilisation description des commandes secondaires

IGNORER: l'utilisateur décide d'ignorer les résultats de la procédure, sans provoquer la fin du programme, celui-ci reste donc chargé.

FIN : l'utilisateur provoque la fin du programme

STATUS SYSID : l'utilisateur s'inquiète de l'état d'une tâche 10070 (pas nécessairement celle contenant les procédures externes)

BATCH, classe, fichier commande, numero de compte : Soumission d'autre tâche au 10070.

NOM : l'utilisateur demande l'identificateur de la tâche contenant les procédures externes.

XI.6. PROGRAMME DE GESTION DE LIGNE 10070.

Nous utiliserons le même mode de description.

Description du mécanisme des boîtes aux lettres sous SIRIS-7 :

Plusieurs processus peuvent communiquer entre eux par des boîtes aux lettres; c'est-à-dire qu'un processus peut déclarer une (ou plusieurs) boîte aux lettres dans laquelle les autres processus peuvent déposer des messages, le propriétaire étant seul autorisé à les lire. De plus, le propriétaire peut se bloquer en attente d'un message dans sa boîte aux lettres. Nous utiliserons ce mécanisme pour faire dialoguer entre eux le programme de gestion de ligne et le programme utilisateur.

Diagramme d'état :

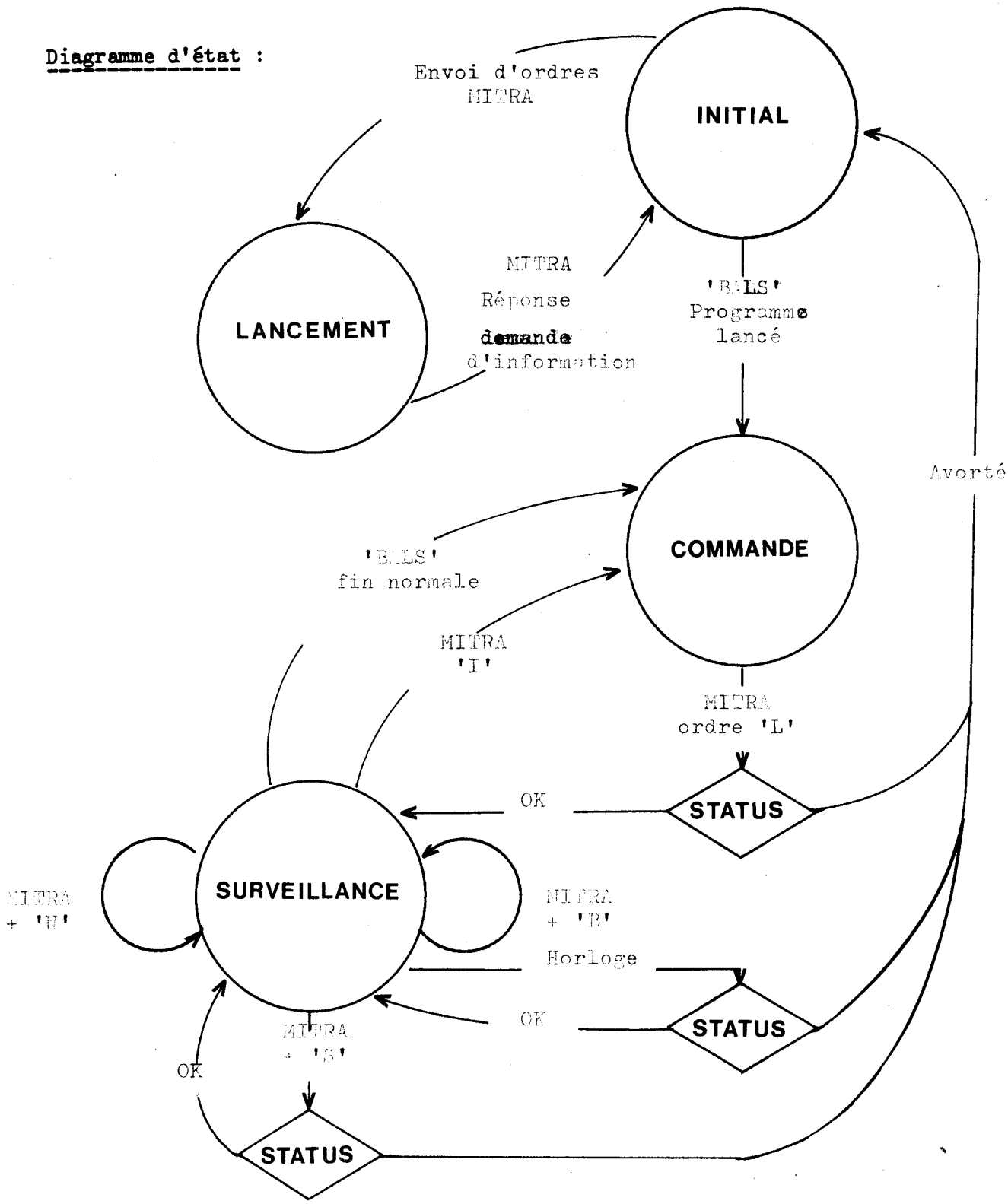


Fig. XI.2.

```
Begin
Integer LIGNE /* identificateur de ligne */, I;
String ORDRE, INFO, SYSID /* identificateur du programme */, A, ORDRE 2
       /* commande secondaire */, B;
       Procedure ATTENDRE (I);
         integer I;
       Begin /* le programme utilisateur n'est pas prêt, il faut attendre
         qu'il soit bloqué, cela se produit soit après le chargement
         (maintenant possible en parallèle des initialisations) soit après
         un ordre IGNORER */
         String A;
         State ATTENTE;
         RECEIVE ('BALS',A); /* on attend un message dans la boîte aux
           lettres */
         on 'BALS' :
           I : = 0;
           return;
         endstate;
         end;
           Ref Procedure DONNEES; /* recupère les données depuis le mitra 15 */
           Ref Procedure RESULTATS; /* envoi des résultats */
           Ref Procedure IMAGES; /* envoi des images */
           OPEN (LIGNE); /* ouverture de la ligne */
           DECLARE ('BALS'); /* declaration de la boite aux lettres */
           State INITIAL;
           I : = 0; /* sémaphore */
           READ (ORDRE); /* attente ordre mitra */
           RECEIVE ('BALS', SYSID) /* attente message Programme utilisateur */
           on MITRA : /* il n'y a peut être pas de programme lancé */
           WRITE ('D'); /* demande d'information */
           enter LANCEMENT;
           on 'BALS' : /* le programme utilisateur est chargé */
           enter COMMANDE;
           endstate;
           State LANCEMENT; /* lancer le programme utilisateur */
```

```
READ (INFØ);
on MITRA:
  If INFØ ¬= 'DEJA LANCE' then
              begin
              BATCH (INFØ,SYSID); /* soumettre le programme au batch */
              WRITE (SYSID); /* donner l'identificateur à l'utilisateur */
              end;
enter INITIAL; /* il n'y a plus qu'à patienter ! */
endstate;
State COMMANDE;
READ (ORDRE);
on MITRA :
if ORDRE = 'F' then /* fin d'exécution du programme utilisateur */
              Begin
              if STATUS (SYSID) ¬ = 'U' then SEND (SYSID 'F');
                 /* envoi de l'ordre */ /* sinon rien à faire car
                 le programme n'est plus là */
              enter INITIAL;
              end;
if SUBSTR (ORDRE, 2,1) = 'D' then DONNEES; /* réception des données */
If I = 0 then ATTENDRE (I); /* synchronisation */
SEND (SYSID, 'L'); /* lancer le programme */
enter SURVEILLANCE; /* il n'y a plus qu'à surveiller */
endstate;
State SURVEILLANCE;
READ (ORDRE 2);
RECEIVE ('BALS', A);
on CLOCK :  /* interruption horloge */
if STATUS (SYSID) = 'U' then
          begin /* avortement */
          WRITE (AVORTE);
          enter INITIAL;
          end;
  idem;
```

```
on MITRA:
   B : = SUBSTR (Ordre 2,1,1);
   IF B = 'S' then
              begin /* demande de status */
              ORDRE 2 : = SUBSTR (Ordre 2,2,4);
              A : = STATUS (ORDRE 2);
              WRITE (A);
              If ORDRE 2 = SYSID & A = 'U' then enter INITIAL; /* programme
                utilisateur avorté */
              end;
  else If A = 'B' then
              begin
                BATCH (ORDRE, 2,1); /* soumettre au batch */
                WRITE (A); /* identificateur */
              end
    else     If A = 'I' then
              begin /* ignorer les résultats */
              I : = 0; /* synchronisation */
              enter COMMANDE;
              end
              else if A = 'F' then
                begin
                if STATUS (SYSID) ¬ = 'U' then SEND (SYSID, 'F');
                enter INITIAL;
                end;
    idem;
on 'BALS' : /* tout s'est bien passé */
    WRITE (A);
    if substr (Ordre, 3,1) = 'R' then RESULTATS;
    if substr (Ordre, 4,1) = 'I' then Images;
    enter COMMANDE;
    endstate;
    end /* du programme */;
```

Références : SGT sous SIRIS 7/8 Procédures systèmes sous SIRIS 7/8
SGF sous SIRIS 7/8 Utilisation des boites aux lettres sous SIRIS 7/8

XI.7. PROCEDURES EXTERNES.

Les procédures externes sont formées d'un programme ayant la forme suivante :

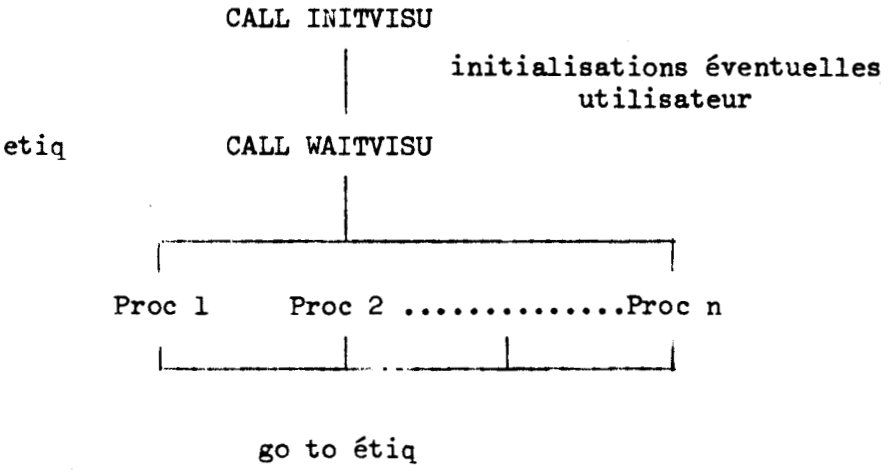


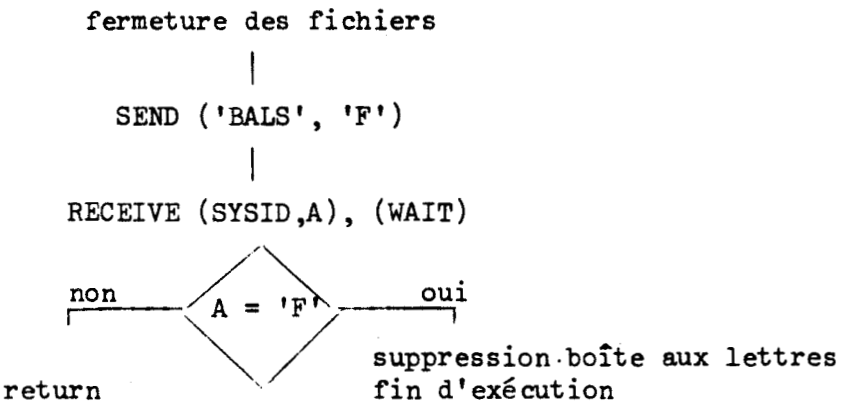
Fig. VI.3

Le programme utilisateur est donc un programme qui boucle la fin étant provoquée par un ordre provenant du MITRA 15.

Procédure INITVISU

contient la déclaration de la boîte aux lettres et le dépôt d'un message dans la boîte aux lettres système.

Procédure WAITVISU



C'est à l'utilisateur de récupérer les données dans le fichier et de remplir ensuite le fichier par les résultats. C'est également à lui qu'incombe la charge de gérer l'aiguillage entre les diverses procédures qu'il peut appeler.

Les initialisations figurant entre les appels de procédures INITVISU et WAITVISU ne sont effectuées qu'une seule fois (au chargement) et en parallèle avec l'exécution du programme d'interactions.

XI.8. MECANISME D'APPEL :

Sur la figure suivante, nous avons représenté le fonctionnement de l'appel d'une procédure externe, en nous limitant à l'exécution de cette procédure, c'est-à-dire que la procédure est supposée chargée et que l'utilisateur attend patiemment la fin d'exécution sans intervenir. Les envois d'ordre ou de renseignements entre les différents processus sont représentés en pointillés, les transitions entre les différents états d'un même processus en traits pleins.

En cas d'avortement durant l'exécution d'une procédure externe, l'utilisateur est prévenu, mais le programme d'intéraction est réactivé dans l'état dans lequel il se trouvait avant l'appel, c'est-à-dire qu'aucune des modifications de données attendues n'a été effectuée. Il pourra donc (si son programme le permet) continuer en séquence.

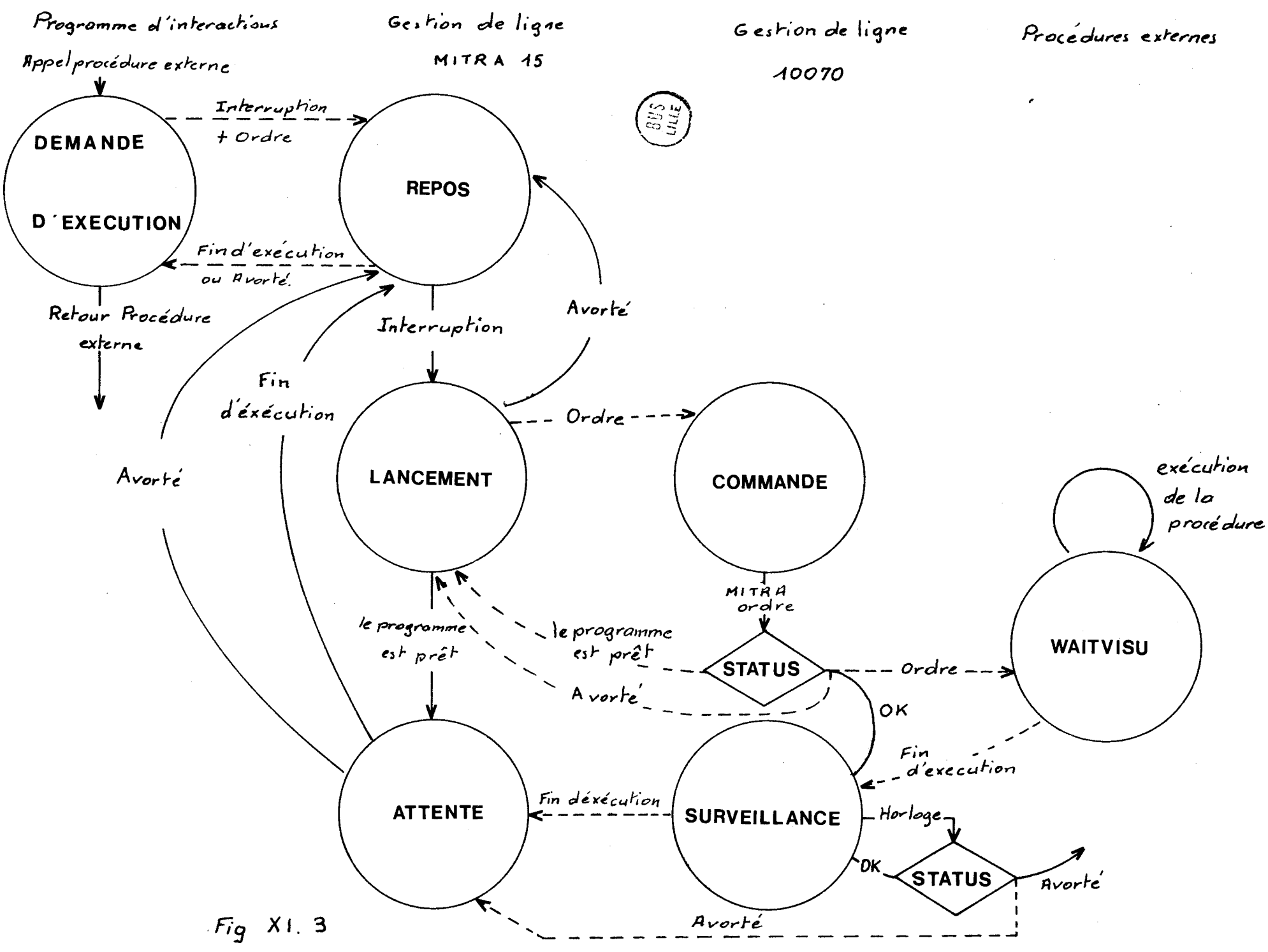

Programme d'interactions
Gestion de ligne
MITRA 15
Gestion de ligne
10070
Procédures externes
Appel procédure externe
DEMANDE
D'EXECUTION
Interruption
+ Ordre
REPOS
Fin d'exécution
ou Avorté.
Retour Procédure
externe
Interruption
Avorté
Fin
d'éxécution
Avorté
LANCEMENT
Ordre
COMMANDE
exécution
de la
procédure
MITRA
ordre
le programme
est prêt
le programme
est prêt
STATUS
Ordre
WAITVISU
Avorté
OK
Fin
d'execution
ATTENTE
Fin déxécution
SURVEILLANCE
Horloge
OK
STATUS
Avorté
Avorté

Fig XI. 3

XI.9. REMARQUES ET AUTRES SOLUTIONS POSSIBLES.

Le mécanisme adopté présente un certain nombre d'inconvénients : Il immobilise une partie des ressources du 10070 alors que le programme utilisateur est en attente et qu'il ne les utilise pas (essentiellement la place mémoire). Il ne gère qu'un seul programme contenant des procédures externes, toujours à cause des ressources de SIRIS 7, car là le gaspillage serait plus considérable encore.

On aurait pu envisager de lancer l'exécution des procédures externes complètement indépendamment du programme temps réel dans la partition time sharing de SIRIS 7 mais les programmes en temps partagé ne peuvent pas utiliser les boîtes aux lettres.

Le mécanisme ne s'applique pas non plus à des procédures interprétées car il n'est généralement pas possible d'appeler dans une telle procédure des sous-programmes écrits dans d'autres langages, les procédures INITVISU et WAITVISU ne seraient donc pas incorporables (excepté si on modifie les interpréteurs).

D'autres solutions pourraient être envisagées si nous disposions d'un autre système sur le calculateur principal. Avec un système effectuant de la pagination, il serait alors possible en gardant le même schéma d'avoir plusieurs programmes.

En fait les seules solutions possibles sont pour éviter la réservation statique de mémoire qui est pénalisante pour le système d'exploitation soit un mécanisme de pagination qui lui permette d'utiliser les ressources du programme utilisateur durant le temps où il est en attente, soit la possibilité d'exécuter les procédures externes comme un usager d'un système temps partagé le ferait.

XII. EXTENSIONS DU SYSTEME.

XII.1. ADJONCTION D'UNE TABLETTE GRAPHIQUE.

- description hardware: La tablette est un moyen d'entrée permettant le relevé de coordonnées.

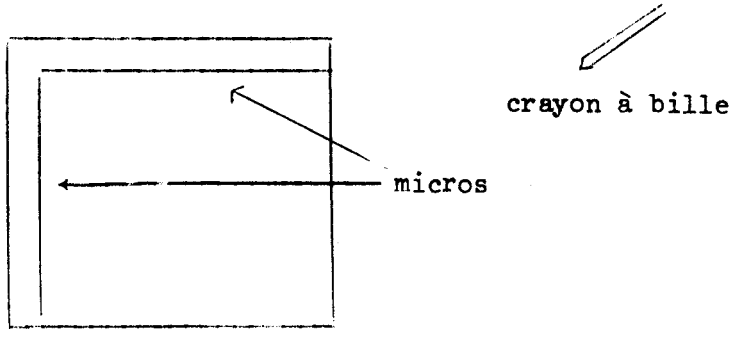


Fig. XII.1

Le crayon envoie des étincelles et les coordonnées du crayon sont calculées en mesurant le temps qui sépare l'envoi de l'étincelle de la réception du bruit sur les micros. La tablette est couplée directement au processeur de la console.

La tablette peut fonctionner suivant plusieurs modes :

- coup à coup pour relever des points isolés,
- commandée par le processeur à raison d'un prélèvement par cycle de rafraîchissement,
- free Run : la tablette fonctionne de façon autonome et envoie une interruption à chaque relevé.

Ces modes sont fixés par l'utilisateur en utilisant le panneau de commande de la tablette. Les coordonnées relevées sont comprises entre 0 et 1024.

Mode d'utilisation.

L'évènement associé à une interruption tablette est l'évènement système 4. L'utilisateur récupérant les coordonnées au moyen de la fonction TABCOOR.

Exemple :

```
Symbol B; integer I;
STATE S1;
TITLE TEXT 'Relevé de coordonnées sur la tablette  et suivi du tracé '
I : = 1
on SYST 4: if I = 1 then begin B : = Move to  TABCOOR -<512,512>;
                                                    I : = 0 end
else B : = B line to tabcoor -  <512,512>;
COPY B at <0,0> ;
idem;
```

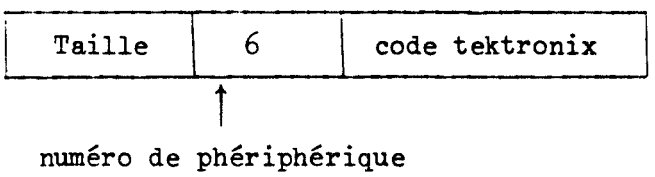
XII.2. ADJONCTION D'UN TUBE A MEMOIRE.

Il est prévu également pour permettre à l'utilisateur d'afficher des images complexes que ne supporterait la console VG 1600 d'ajouter un tube à mémoire Tektronix 4014.

Caractéristiques principales:

taille de l'écran 4096×3120 points visibles
38×30,6 cm.

Il n'est pas prévu actuellement de générer des symbôles sur ce tube. Les images sont donc générées par les procédures externes sous la forme suivante :



Les instructions utilisées pour afficher ce type d'image sont les mêmes que pour les images CIT, le système connaissant le type de l'image.

DISPLAY affiche l'image sans effacer l'écran
(possibilité de superposer plusieurs images)
DELETE efface tout l'écran

l'instruction light est inopérante de même que l'évènement désignation d'une image tektronix.

Un évènement système (SYST 5) et une fonction (Tekcoor) permettent de relever les coordonnées d'un point sur l'écran au moyen du joystick.

XII.3. ACCES AU RESEAU CYCLADES

Il est envisagé de permettre à l'utilisateur via le réseau cyclades d'exécuter ses procédures externes sur un autre calculateur que le 10070. La solution qui nous paraît actuellement la plus aisée à mettre en oeuvre est la suivante :

- remplacer le programme contenant les procédures externes par la station de transport cyclades, le programme temps réel étant modifié pour pouvoir fournir les informations nécessaires à cette station de transport.

Cette solution nous paraît être la meilleure car elle évite de devoir coupler le calculateur satellite aux MITRAS cyclades qui assurent les liaisons entre les différents ordinateurs et d'implanter sur le calculateur satellite les programmes nécessaires pour l'interface.

XII CONCLUSION

Le système GIPSY que nous venons de décrire vérifie bien les principaux objectifs que nous nous étions fixés :

1. indépendance vis à vis des applications.

   Le système n'est pas orienté vers un type particulier d'applications.

   Par contre la programmation des applications est facilitée par

   - la séparation des tâches : un programme principal écrit en langage de specification des intéractions qui décrit le dialogue homme-machine
   - les procédures externes s'exécutant dans le calculateur principal.

   Ceci permet une mise au point séparée des procédures de calcul et de dialogue et facilite les mises à jour

   - la puissance des outils fournis.

   Les outils fournis par le système sont de haut niveau : l'utilisateur n'a pas à se préoccuper ni de la gestion de la liste d'affichage ,i de la récupération des interruptions.

2. indépendance vis à vis du terminal graphique.

   Tout programme d'application doit pouvoir s'exécuter sur l'un quelconque des terminaux graphiques disponibles, certaines fonctions pouvant être ignorées si elles n'existent pas sur le terminal considéré (exemple : couleur pour le tektronix) et ceci sans modification du programme source.

3. possibilité d'utiliser le calculateur satellite seul.

   Ceci est déjà possible, mais pour donner à l'utilisateur les mêmes moyens que sur l'ensemble du système. Il est prévu de réaliser le transport du FORTRAN 3D sur le MITRA. Les procédures externes seraient alors situées également dans le MITRA.

Le principal problème reste cependant le temps de réponse. La réponse aux interventions de l'utilisateur est immédiate, tout se déroulant au niveau du MITRA 15. Par contre, lors de l'appel de procédures externes, le temps de réponse dépend fortement du système d'exploitation du calculateur principal.

Dans le cas particulier de SIRIS 7, on peut relever les inconvénients suivants :

- le lancement de l'exécution dans le batch pénalise l'utilisateur du système par rapport aux usagers du time-sharing
- l'occupation permanente de la mémoire par les procédures externes gène profondément le système SIRIS 7.

Les améliorations prévues en modifiant l'interface sont :

- ne pas attendre la fin de l'exécution de la procédure externe dans le cas où celle ci ne revoie aucun résultat
- transférer les données et les résultats sous forme binaire via les boites à lettres (ce qui évite l'utilisation de fichiers mais limite le nombre de paramètres.

Bibliographie

[1] W.M. NEWMAN
Display Procedures
Comm. ACM Vol 13 n° 12 (Dec. 70).

[2] J. GROS, E. SALTEL
Extension de la notion de procédure pour la génération d'images sur des terminaux graphiques
Congrès AFCET, Grenoble (Nov. 72).

[3] E. SALTEL
METAVISU III Manuel d'utilisation
Note de travail METAVISU A/004 (Fev. 74).

[4] J.A. FELDMAN, P.D. ROVNER
An ALGOL associative based language
Comm. ACM Vol 12, n° 1 (March 71).

[5] P. BOULLIER, P. JANCENE
Langage pour la construction et la manipulation d'une base de données associative
Congrès AFCET Grenoble (Nov. 72).

[6] R. WILLIAMS
A survey of data structure for computer graphics systems
Computing Surveys Vol 3, n° 1 (March 71).

[7] W.M. NEWMAN
A system for interactive graphical programming
Proceedings AFIPS 1968 SJCC

[8] N.E. WISEMAN, J.O. MILES
A ring structure processor for a small computer
Computer Journal Vol 10 n° 4 (Feb. 68)

[9] C.A. LANG, J.C. GRAY
A.S.P. a ring implemented associative structure package
Comm. ACM Vol 11 n° 8 (Aug. 68).

[10] D.L. CHILDS
Description of a set theoretic data structure
Proceedings AFIPS, 1968 FJCC Vol 33.

[11] C.I. JOHNSON
Principles of interactive systems
IBM System Journal N° 384 (1968).

[12] IBM SYSTEM/360
Operating System Graphic Programming Services for FORTRAN IV
C 27 - 6932 - 1

[13] N.E. WISEMAN, C.J. CHENEY, M. ETHERTON, J.O. MILES and M.U. LEMKE
RAINBOW - A multi-purpose CAD System
Software Practice and Experience Vol 2 pp 359 - 372 (1972)

[14] C. CHRISTENSEN, E.N. PINSON
Multi functions graphics for a large computer system
Proceedings AFIPS 1967 FJCC Vol 31

[15] W.M. NEWMAN, H. GOURAUD, D.R. OESTREICHER
A programmer's guide to PDP 10 EULER
Computer Science Technical Report Univeristy of UTAH (June 1970).

[16] I. COTTON, F.S. GREATOREX
Data structures and techniques for remote computer graphics
Proceedings AFIPS 1968 FJCC Vol 33

[17] W.M. NEWMANN
An experimental display programming language for the PDP 10 Computer
Computer science technical report Univ. of UTAH (July 70).

[18] A. DUCROT, A. LEMAIRE, F. PRUSKER
METAVISU un système graphique interactif
Rapport LABORIA n° 36 (Nov. 1973)

[19] C. BETOURNE, J. BOULENGER, J. FERRIE, C. KAISER, S. KRAKOWIAK, J. MOSSIERE
Process management and resource sharing in the multi access system ESOPE
Comm. ACM Vol 13 n° 12 (Dec. 70).

[20] F. BLAIZOT, L. BLAIZOT, B. LORHO, M. VATOUX
Organisation du compilateur interpréteur CPL/1
Congrès AFCET Paris (Sept. 70).

[21] P. BOULLIER, J. GROS, P. JANCENE, A. LEMAIRE, F. PRUSKER, E. SALTEL
METAVISU A general purpose graphic system
IFIP working conference of graphic languages, Vancouver (May 1972)

[22] W.M. NEWMAN, R. SPROULL
Principles of interactive computer graphics
Mc Graw Hill (1973)

[23] Groupe graphique de l'AFCET
Rapport sur les principes de conception et d'organisation des systèmes graphiques
Journées graphiques AFCET/IRIA (dec 73).

[24] W.M. WEWMAN
A device independent system
Journées graphiques AFCET/IRIA (dec 73).

[25] P.A. WOODSFORD
The design and the implementation of the GINO 3D graphic software package
Software Practice and Experience Vol 12 (1971).

[26] E. SALTEL
Utilisation de la génération dynamique et des procédures graphiques dans le software METAVISU III
Thèse de 3ème cycle Univ. PARIS VI (mai 1974).

[27] I.W. COTTON
Languages for graphic attention-handling
Computer graphics (1970).

28 I.E. SUTHERLAND
SKETCHPAD a man machine graphical communication system
Proceeding ACM Spring Joint Conference (1963)

ANNEXE

Présentation sommaire de l'extension graphique de FORTRAN

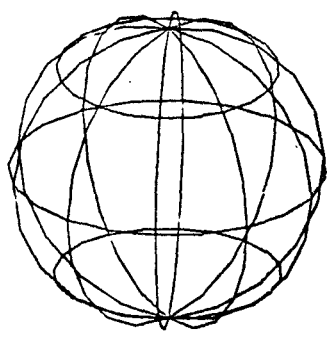
Notion de procédure graphique

Dans le langage de programmation, la notion de procédure est utilisée pour définir une suite d'actions qui s'appliquent sur des paramètres formels. Le corps de la procédure est donc un modèle paramétrable, appelé en différents endroits du programme avec des arguments réels.

Une image simple est décrite par une suite d'instructions graphiques élementaires (ligne, positionnement, texte). Cette suite ayant une certaine unité sémantique, il est intéressant de pouvoir la décrire et la manipuler comme un tout. C'est-à-dire, pouvoir décrire l'image comme un modèle indépendamment du contexte où elle sera utilisée.

Il existe donc une certaine analogie entre la notion de procédure classique et la notion d'image. En conséquence, nous considérons que la description d'une image ne puisse se faire que dans le corps d'une procédure et non pas n'importe où dans un programme. La paramétrisation de cette image est alors assurée à l'appel de la procédure par des options graphiques spécifiant quelles sont les opérations à effectuer sur l'image décrite. Par exemple à partir d'une subroutine décrivant un polygone régulier dans le plan xoy, la description d'une sphère de rayon 1 se fait de la manière suivante :

```
1     C
2           SUBROUTINE SPHERE
3           REAL PI
4           DATA PI/3,141592653/
5           DO 1 I=0,7,1
6           CALL POLYGONE(1.,2.*PI,20) ROTY(I*PI/8.) ;
7     1      CONTINUE
8           DO 2 XI=-1.,1.,1./5.
9           CALL POLYGONE(1.,2.*PI,16) AT(0.,XI,0.) ROTX(PI/2.)
10           SCALE(SQRT(1.-XI**2),1.,SORT(1.-XI**2)) ;
11    2      CONTINUE
12          RETURN
13          END
```



ELEMENT SIMPLES DE GENERATION

Instruction de positionnement absolu ou relatif avec ou sans marquage

```
                    TO      ( { x,y[, z] } )
MOVE ∧ Σ            OF      ( { dx,dy[, dz] })      ;
                    MARK        (ch, n, s, t)
```

- L'option TO permet de positionner le faisceau au point de coordonnées absolues x, y [,z].
- L'option OF déplace le faisceau d'un vecteur dx, dy [,dz] à partir de la position courante xc, yc [,zc] pour atteindre le point de coordonnées xc + dx, yc + dy [,zc + dz].
- L'option MARK précise la chaîne de caractères qui sera affichée à chacune des positions indiquées par les options TO et OF suivantes.
  Ceci jusqu'au ';' ou jusqu'à la prochaine option MARK.

Instruction de tracé de lignes ou de vecteurs

```
                    TO      {x, y [,z]}
                    OF      {dx, dy  [,dz]}
LINE ∧ Σ            FROM    {x, y [,z]}          ;
                    BY      {dx, dy [,dz]}
                    WITH      (t)
```

- L'option FROM précise un positionnement absolu du faisceau au point de coordonnées x,y [,z]. Elle est identique à une instruction MOVE TO(x,y [,z]) ;
- L'option BY précise un déplacement relatif du faisceau d'un vecteur dx, dy [,dz] à partir de la position courante. Elle est identique à une instruction MOVE OF(dx, dy [,dz]) ;
- L'option TO permet de tracer un segment depuis la dernière position atteinte jusqu'au point de coordonnées absolues x, y [, z] ;
- L'option OF permet de tracer à partir de la dernière position atteinte xc, yc [,zc] un vecteur de composante dx, dy [,ds] , c'est-à dire jusqu'au point de coordonnées xc + dx, yc + dy [,zc + dz]
- L'option WITH précise le mode de tracé des segments qui suivent jusqu'au ';' ou jusqu'à la prochaine option WITH rencontrée.

OPTIONS GRAPHIQUES

```
CALL        nom [(Σarguments)]

[ Σ { AT    ({x, y [,z]}) } ]
    { BY    ({x, y [,z]}) }

[ Σ { DIM       (exp)
    { LIMITS    (exp)
    { PLACE     (exp)
    { SIZE      ({dx, dy, dz})
    { WITHIN    x, y, dx, dy
    {           x, y, z, dx, dy, dz
    { ONTO      x, y, dx, dy
    {           x, y, z, dx, dy, dz
    { ROT       ({α, β, γ})
    { ROTX      (β)
    { ROTY      (γ)
    { ROTZ      (α)
    { SCALE     ({a, b, c})
    { TRANSLA   ({a, b, c})
    { TRANSFO   (exp)
    { EXEC      (exp)
    { FORMAT    (exp)
    { PERSPECT  ({xo, yo, zo, x, y, z, f})
    { PERSPECT1
    { LIGHT     ({i,d,co})
    { APPLY     (nom)
    { AS        (n)
```

AT et BY définissent l'emplacement de l'origine du système de coordonnées de l'image décrite dans l'espace appelé.

DIM indique le nombre de dimensions de la procédure appelée. Si l'option DIM n'apparait pas, le nombre de dimensions est pris égal à 2.

Les paramètres optionnels z ou dz qui apparaissent dans la description de la syntaxe des instructions graphiques, deviennent obligatoires si le contexte de l'instruction dans laquelle ils apparaissent est à 3 dimensions sinon, ils ne sont pas interprétés et deviennent donc optionnels.

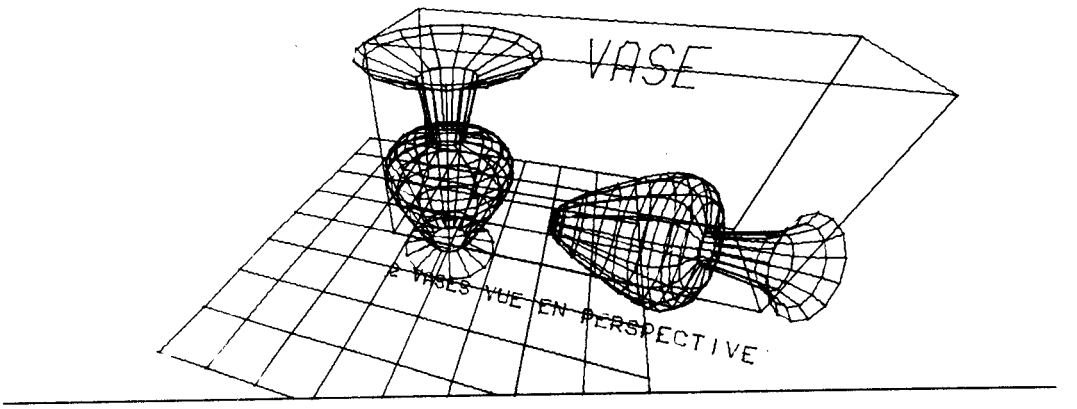
Définition d'un masque et d'une fenêtre

Dans un espace donné, l'image peut être un dessin très complexe et de grande dimension. L'utilisation d'un écran cathodique implique une limitation de l'image due à la dimension de l'écran et à la nécessité de rafraichir l'image.

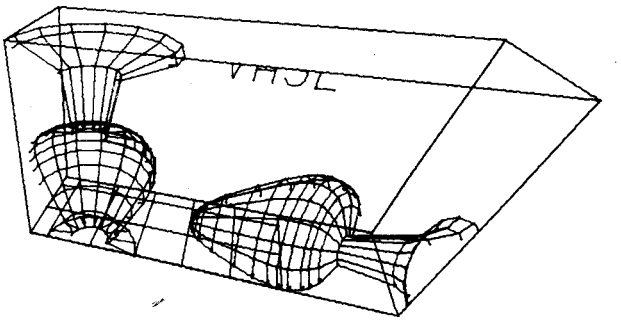
De plus, à un instant donné, l'utilisateur peut ne vouloir travailler que sur une partie de son image générale. Pour cela, le système fournit un outil de découpage permettant de sélectionner une région rectangulaire définie dans l'espace appelé (le masque) et de la projeter dans une région rectangulaire de l'espace appelant (la fenêtre).

Exemple de découpage en 3D

Vue des 2 vases avant découpage :
Le parallélépipède rectangle par lequel ils seront découpés a été matérialisé.



Vue des 2 mêmes vases après découpage :
L'option LIMITS matérialise la boite par laquelle les vases sont découpés.



Transformation de l'image dans l'espace appelé.

Les options suivantes permettent de définir une suite de transformations affines qui modifient le dessin décrit dans la procédure appelée. Soient x, y, z, les coordonnées d'un point avant transformation et x', y', z' après transformation dans l'espace appelé.

Les transformations sont effectuées de gauche à droite dans l'appel de la procédure. En fait, le système calcule à l'exécution la matrice de transformation résultante qui sera utilisée avant l'opération de découpage.

Facteur d'échelle

SCALE effectue $x' = ax$
$y' = by$
$z' = cz$

Rotations

Autour de l'axe oz. : ROTZ(α)
Autour de l'axe oy. : ROTY(γ)
Autour de l'axe ox. : ROTX(β)

Translations

TRANSLA effectue $x' = x + a$
$y' = y + b$
$z' = z + c$

Matrice de transformation

TRANSFO : Cette option permet de spécifier une matrice de transformations en coordonnées homogènes.

Perspectives

PERSPECT : Cette option permet d'obtenir l'image vue par un observateur se trouvant au point xo, yo, zo et regardant le point x, y, z, avec un appareil photo ayant une distance focale f.

FORMAT(ch) : où ch est une chaîne de caractères. Le préprocesseur génère : CALL ch. Donc ch doit être un identifieur de subroutine suivi ou pas de ses paramètres entre parenthèses.

Transformation quelconque

APPLY(nom) : Transformation ponctuelle quelconque par la subroutine 'nom'.

Caractéristiques de tracé pour l'image appelée

LIGHT(i, d, co)

i indique l'intensité du faisceau

d indique le clignotement du faisceau

co indique la couleur.

Ces caractéristiques ne s'appliquent qu'à la procédure appelée. Au retour de l'appel, le système revient aux caractéristiques valables avant l'appel. Ceci se situe dans la logique de la neutralité graphique des appels de procédures.

Désignation d'une image

AS(n)

n est une expression à résultat entier tel que $|n| < 2^{16} - 1$

Cette option si le périphérique le permet, affecte un nom à l'image appelée et la rend sensible au "crayon optique" pour une désignation en interaction. Le nom de l'image est le nombre entier.

AUTRES INSTRUCTIONS GRAPHIQUES

Définition du périphérique

GRAPHIQUE(exp) : cette instruction permet de définir le type du périphérique pour lequel l'image sera générée. En cours de programme, elle permet de changer dynamiquement de périphérique. Elle ne peut évidemment être exécutée qu'en niveau 1 de procédure graphique.

Variable externe LEVEL

Nous avons défini précédemment le niveau graphique d'une procédure. La variable LEVEL représente à tout instant le niveau graphique de la procédure dans laquelle elle apparait. Sa valeur minimum est 1. C'est la valeur du niveau à l'extérieur de la procédure graphique de niveau le plus bas.

Calcul de l'homologue d'un point dans un autre espace

TRANSPAC(x,y,z) [FROM(n) TO(m)] ;

Cette instruction permet de calculer l'homologue du triplet x,y,z, interprété comme représentant les coordonnées d'un point de l'espace de niveau graphique n, dans l'espace de niveau graphique m. Les valeurs des coordonnées des points homologues sont réaffectées aux variables x,y,z, par reflet.

La relation $1 \leq n \leq$ LEVEL et $1 \leq m \leq$ LEVEL, doit toujours être vérifiée

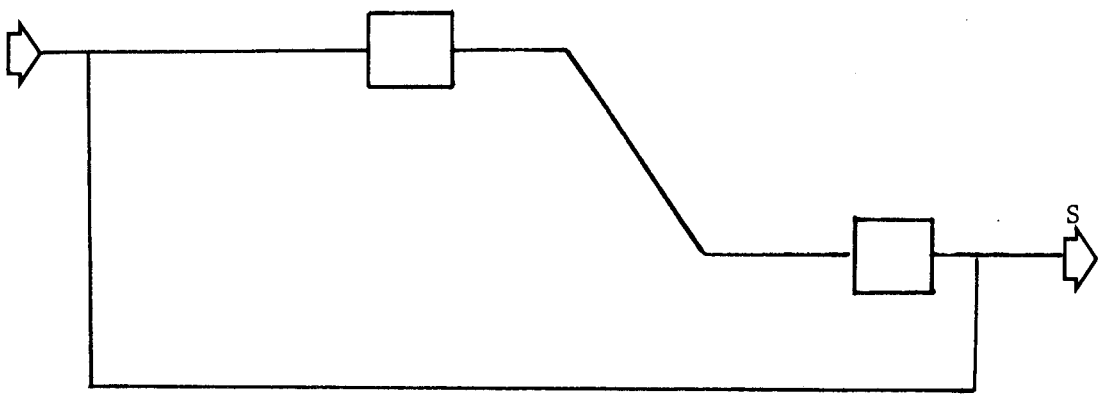
Exemple de programme d'interaction

Simulation d'un réseau de files d'attente.

L'exemple ci-dessous ne comporte que la partie édition du réseau. Le réseau comporte trois types d'éléments :

- les entrées
- les servants
- les sorties S

reliés entre eux par des liaisons



Exemple de réseau obtenu

Le programme permet d'ajouter ou de retrancher des éléments au réseau et crée la structure de données correspondantes.

```
BEGIN
EXTERNAL IMAGE(INTEGER,FRAME);
EXTERNAL SUPPRESS(INTEGER,ITEM,INTEGER,INTEGER,FRAME);
EXTERNAL SUPPRESS1(INTEGER,STRING,INTEGER,INTEGER,FRAME);
PAIR LIST L1,L2,L3;
FRAME RESEAU;
SYMBOL ENTREE,SERVANT,TORTIE,AST;
INTEGER I,J;
     PROCEDURE AJOUTE(I); /* AJOUTE UN ELEMENT AU RESEAU */
     INTEGER I;
     EXTERNAL PROCEDURE ELEMENT(INTEGER,PAIR,STRING);
     BEGIN
     STATE AJOU;
     TITLE TEXT 'AMENER L"ELEMENT A LA POSITION
          DESIREE' MOVE TO <0,-10>
          TEXT 'LE NOMMER';
     EDIT;
     TRACKING WITH HITSYMBOL;
     ON KEY F1:
          CALL ELEMENTS(I,CROSSCOOR,INTEXT);
          RETURN;
     ENDSTATE;
     END;
     SYMBOL PROCEDURE AFFICHER(LS);
     /*CALCULE LE SYMBOLE DE LA LIAISON
        AU COURS DE SA CONSTRUCTION */
     PAIR LIST LS;
     BEGIN
     SYMBOL CROQUIS,INTEGER I;
     CROQUIS:=MOVE TO LS[1];
     FOR I:=1 UNTIL COUNT(LS) DO
     CROQUS:=CROQUIS LINE TO LS[I];
     RETURN CROQUIS;
     END;
     INTEGER PROCEDURE LEPLPRO(K,POINTRAV);
     /* RECHERCHE DANS LA LISTE POINTRAV LE
        POINT LE PLUS PROCHE DE K */
     PAIR K;PAIR LIST POINTRAV;
     BEGIN
     REAL D,DMIN;INTEGER J,JMIN;
     DMIN:=1000000.;
     FOR I:=1 UNTIL COUNT(POINTRAV) DO
          BEGIN
          D:=DISTANCE(K,POINTRAV[J]);
          IF D<DMIN THEN
               BEGIN
               DMIN:=D;
               JMIN:=J;
               END;
          END;
     RETURN JMIN;
     END;
```

```
SYMBOL PROCEDURE FLECHE(K);
 /*CALCULE UNE FLECHE DONT L'ORIENTATION VARIE
   SUIVANT LE QUADRANT OU SE TROUVE K *
  /
BEGIN
INTEGER I;
IF FIRST K<0 THEN N:=1 ELSE N:=2;
IF SECOND K>=0 THEN N:=N+2;
CASE N
      OF RETURN LINE OF <4,10> OF <0,-4>
            OF <14,22> OF <0,-10> OF <10,0>
            OF <-22,-14> OF <4,0> OF <-10,-4>
      OR RETURN LINE OF <-4,10> OF <0,-4>
            OF <-14,22> OF <0,-10> OF <-10,0>
            OF <22,-14> OF <-4,0> OF <10,-4>
      OR RETURN LINE OF <-4,-10> OF <0,4>
            OF <14,-22> OF <0,10> OF <10,0>
            OF <-22,14> OF <-4,0> OF <10,4>
      OR RETURN LINE OF <-4,-10> OF <0,4>
            OF <-14,-22> OF <0,10> OF <-10,0>
            OF <22,14> OF <-4,0> OF <10,4>;
END;
INTEGER PROCEDURE RELIER(L2,LS,SENS,MIDE);
/* TERMINE LA LIAISON */
PAIR LIST L2,LS; INTEGER MIDE,SENS;
BEGIN
PAIR LIST POINTRAV; INTEGER I,K;
SYMBOL CUILA;
POINTRAV:=L2;
I:=COUNT(LS);
STATE S;
TITLE TEXT 'LE POINT DESIGNE EST IL LE BON?';
K:=LEPLPRO(LS[I],POINTRAV);
IF K=0 THEN RETURN 0;
CUILA:=FLECHE(POINTRAV[K]);
COPY CUILA AT POINTRAV[K] LIGHT 1,1,2;
ON TEXT 'NON':
      REMOVE K FROM POINTRAV;
      ENTER S;
ON TEXT 'OUI':
```

```
      CASE MIDE
           OF LS[2]:=POINTRAV[K];
           OR BEGIN
                IF I=2 THEN
                     BEGIN
                     IF SENS =1 THEN
                          LS[2]:=<FIRST LS[2],
                                  SECOND POINTRAV[K]>;
                     ELSE LS[2]:=<FIRST POINTRAV[K]
                                 ,SECOND LS[I]>;
                     INSERT POINTRAV[K] TAIL LS;
                     END
                ELSE BEGIN
                     LS[I]:=POINTRAV[K];
                     I:=I-1;
                     IF SENS =1 THEN
                          LS[I]:=<FIRST POINTRAV[K]
                                 ,SECOND LS[I]>;
                     ELSE LS[I]:=<FIRST LS[I],
                                 ,SECOND POINTRAV[K]>;
                     END;
                END;
           END;
      RETURN 1;
ENDSTATE;
END;
INTEGER PROCEDURE CRAENF(L2,LS,SENS,MIDE);
PAIR LIST L2,LS;INTEGER SENS,MIDE;
BEGIN
STATE SPUP;
TITLE TEXT 'QUE FAIRE?';
ON TEXT 'CONTINUER':
      RETURN 0;
ON TEXT 'RELIER AU POINT LE PLUS PROCHE':
      RETURN RELIER(L2,LS,SENS,MIDE);
ENDSTATE;
END;
```

```
PROCEDURE LHV(LS,SENS);
/* LIGNES HORIZONTALES ET VERTICALES */
PAIR LIST LS,INTEGER SENS;
BEGIN
INTEGER ARRAY CHANGE[1:2];
INTEGER K,I,DELTA1,DELTA2;
I:=COUNT(LS);
CHANGE[1]:=2;CHANGE[2]:=1;
K:=1;
WHILE K¬=0 DO
      BEGIN
      IF SENS =1 THEN
            BEGIN
            SECOND LS[I]:=SECOND CROSSCOOR;
            DELTA1:=ABS(FIRST LS[I]-FIRST CROSSCOOR);
            DELTA2:=ABS(SECOND LS[I-1]-SECOND CROSSCOOR);
            END
      ELSE BEGIN
            FIRST LS[I]:=FIRST CROSSCOOR;
            DELTA1:=ABS(SECOND LS[I]-SECOND CROSSCOOR);
            DELTA2:=ABS(FIRST LS[I-1]-FIRST CROSSCOOR);
            END;
      IF DELTA1>=32 THEN
            BEGIN
            IF DELTA2>=32 THEN
                  BEGIN
                  INSERT CROSSCOOR TAIL LS;
                  SENS:=CHANGE[SENS];
                  K:=0;
                  END
            ELSE K:=0;
            END
      ELSE BEGIN
            IF DELTA2>=32 THEN
                  BEGIN
                  SENS:=CHANGE[SENS];
                  IF I=2 THEN LS[2]:=LS[1]
                  ELSE BEGIN
                        REMOVE TAIL LS;
                        I:=I-1;
                        END;
                  END;
            END;
      END;
END;
```

```
PROCEDURE RELIER(L2,AST,RESEAU);
PAIR LIST L2;SYMBOL AST;FRAME RESEAU;
BEGIN
EXTERNAL PROCEDURE LIAISON(INTEGER,INTEGER,INTEGER,PAIR
     LIST ,FRAME);
INTEGER SENS,MIDE;PAIR DEPART;
PAIR LIST LS;SYMBOL CROQUIS;
SENS:=1;
STATE UN;
TITLE TEXT 'DESIGNER L"ELEMENT DE DEPART'
ON PEN AST:
     DEPART:=HITCOOR+<45,0>
     DELETE AST;
     INSERT DEPART TAIL LS;
     INSERT DEPART TAIL LS;
     ENTER DEUX;
ENDSTATE;
STATE DEUX;
TITLE TEXT 'CHOISIR LE MODE DE TRACE';
ON TEXT 'NORMAL':
     MIDE:=1;
     ENTER TROIS;
ON TEXT 'LIGNES HORIZONTALES ET VERTICALES':
     MIDE:=2;
     ENTER TROIS;
ENDSTATE;
STATE TROIS;
CROSS AT DEPART;
ON SYST 1: /* CYCLE */
     IF CROSSCOOR¬=DEPART THEN
          BEGIN
          IF MIDE=1 THEN LS[2]:=CROSSCOOR
          ELSE LHV(LS,SENS);
          CROQUIS:=AFFICHER(LS);
          COPY CROQUIS AT <0,0> LIGHT 0,0,3;
          END;
     IDEM;
ON SYST 2: /*VALIDATION TRACKING*/
     IF CRAENF(L2,LS,SENS,MIDE)=0
          THEN BEGIN
          DEPART:=LS[COUNT(LS)];
          ENTER DEUX;
          END;
     CROQUIS:=AFFICHER(LS);
     COPY CROQUIS AT <0,0> LIGHT 0,0,3;
     CALL  LIAISON(4,COUNT(LS),LS,*RESEAU);
     DISPLAY RESEAU;
     RETURN;
ENDSTATE;
END;
```

```
STATE DEBUT;
/* DEBUT DU PROGRAMME D'INTERACTION*/
TITLE TEXT 'QUE VOULEZ VOUS FAIRE':
ON TEXT 'CALCUL':IDEM;
ON TEXT 'MODIFICATIONS':ENTER CHOIX;
ON TEXT 'FIN':EXIT;
ENDSTATE;
STATE CHOIX;
TITLE TEXT 'CHOISIR LA MODIFICATION';
ON TEXT 'AJOUTER':ENTER CHOIX2;
ON TEXT 'SUPPRIMER':ENTER SUP;
ON TEXT 'FIN MODIFS':ENTER DEBUT;
ENDSTATE;
STATE SUP;
TITLE TEXT 'DESIGNER L"ELEMENT A SUPPRIMER'
     MOVE TO <0,-10> TEXT 'AU CRAYON OU EN LE NOMMANT';
EDIT;
ON KEY FM:
     CALL SUPPRESS1(5,INTEXT,*I,*J,*RESEAU);
     DISPLAY RESEAU;
     CASE J
     OF REMOVE I FROM L1
     OR REMOVE I FROM L2
     OR REMOVE I FROM L3;
     ENTER CHOIX;
ON PEN RESEAU:
     CALL SUPPRESS(6,HITNAME,*I,*J,*RESEAU);
     DISPLAY RESEAU;
     CASE J
     OF REMOVE I FROM L1
     OR REMOVE I FROM L2
     OR REMOVE I FROM L3;
     ENTER CHOIX;
ENDSTATE;
```

```
STATE CHOIX2;
TITLE TEXT 'QUE VOULEZ VOUS AJOUTER?';
SERVANT:=LINE OF <7,0> OF <0,15> OF <30,0>
    OF <0,-30> OF <0,15> LINE FROM <37,0> OF <8,0>;
ENTREE:=LINE OF <0,7> OF <30,0> OF <0,7> OF <15,14>
     OF <-15,-14> OF <0,7> OF <-30,0> OF <0,7>;
SORTIE:=ENTREE MOVE OF <15,15> TEXT 'S';
COPY SERVANT AT <0,500>;
COPY ENTREE AT <0,490>;
COPY SORTIE AT <0,480>;
ON PEN ENTREE:
     DELETE SORTIE;
     DELETE SERVANT;
     AJOUTER(1);
     CALL IMAGE(7,*RESEAU);
     DISPLAY RESEAU;
     ENTER CHOIX2;
ON PEN SORTIE:
     DELETE ENTREE;
     DELETE SERVANT;
     AJOUTER(3);
     CALL IMAGE(7,*RESEAU);
    DISPLAY RESEAU;
     ENTER CHOIX2;
ON PEN SERVANT:
     DELETE SORTIE;
     DELETE ENTREE;
     AJOUTER(2);
     CALL IMAGE(7,*RESEAU);
     DISPLAY RESEAU;
    ENTER CHOIX2;
ON TEXT 'LIAISON':
     AST:=TEXT '*';
     FOR I:=1 UNTIL COUNT(L1) DO
     COPY AST AT L1[I] LIGHT 0,0,2;
     FOR I:=1 UNTIL COUNT (L2) DO
     COPY AST AT L2[I] LIGHT 0,0,2;
     LIER(L2+L3,AST,RESEAU);
     ENTER CHOIX2;
ON TEXT 'FIN': ENTER CHOIX;
ENDSTATE;
END
```