Institut National Polytechnique de Lorraine

Centre de Recherches Informatique de Nancy
(C.R.I.N)

THESE

Présentée pour l'obtention de grade de
DOCTEUR INGENIEUR EN INFORMATIQUE

par

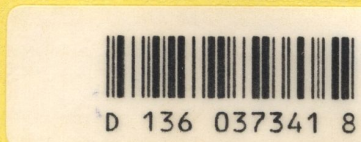
Ahmed CHTAYTI


# ETUDE D'UN SYSTEME D'AIDE A LA PROGRAMMATION D'APPLICATIONS CONVERSATIONNELLES : SPECIFICATION ET REALISATION D'UNE MAQUETTE



soutenue publiquement le 08 Novembre 1984
devant la commision d'examen

Président : J.P. FINANCE
Examinateurs : D. COULON
G. BRISSON
A. SCHAFF
R. STUTZMANN



Je tiens à remercier ici:

Monsieur J.P. FINANCE, Professeur à l'Université de Nancy I de l'honneur qu'il me fait en présidant ce jury. Je lui suis très reconnaissant pour son soutien, ses conseils et l'intérêt qu'il a porté à mon travail.

Monsieur A.SCHAFF, Maître-Assistant à l'université de Nancy I pour son dévouement à mon égard, son aide, ses conseils et ses encouragements.

Monsieur G. BRISSON, Ingenieur à la Direction des Etudes et Recherches d'EDF de m'avoir initié à ce travail et d'avoir accepté de juger cette thèse.

Monsieur D. COULON, Professeur à l'INPL pour l'intérêt qu'il porte à ce travail et pour sa participation au jury.

Monsieur R. STUTZMANN, Maître-Assistant à l'IUT de Strasbourg d'avoir accepté de juger ce travail et pour sa participation au jury.

Je tiens à remercier aussi Mr. P. BOCCARD, Ingenieur CNAM pour sa collaboration dans la réalisation du système.

Enfin que tous les membres de l'équipe Programmation soient remerciés pour leur acceuil et les discussions intéressantes que j'ai eu avec avec eux.

# PLAN - SOMMAIRE.

# INTRODUCTION.

Dans tous les domaines informatiques l'utilisation de moyens onversationnels se généralise de plus en plus. Si cette iffusion est, sur le plan du système d'exploitation (accès au ystème, temps de réponse..) assez bien maitrisée, il n'en va pas e même pour la conception et la réalisation des applications onversationnelles. Les difficultés se sont accrues ces ernières années du fait des perfectionnements techniques des onsoles de visualisation : du matériel "ligne-à-ligne" télétypes, IBM 2741, ...) on est passé à du matériel ravaillant, en option ou en standard en mode pleine-page (ou ode bloqué) ; c'est ce qu'on appelle le dialogue "pleine-page". 'introduction du dialogue "pleine-page" ajoute au problèmes des pplications conversationnelles, qui sont loin d'être complète- ent résolus (en particulier au niveau de l'ergonomie) un autre egré de difficulté. EIle nécéssite des modifications importantes es systèmes informatiques matériels et logiciels.
n effet, d'une part elle nécessite la mise au point d'un angage et des outils de description du dialogue. D'autre art, elle nécessite le développement de nouvelles méthodes de rogrammation adaptées à ce genre de problème.

Les outils proposés jusque là par les constructeurs sont 'accès difficile et sont réservés à des spécialistes. Ils ne ermettent pas de réaliser facilement des maquettes de ialogue servant de support entre informaticien et utilisateur inal ou pour la vérification à priori des divers niveaux de pécification d'un logiciel d'application.

L'objectif de ce travail est d'étudier les problèmes posés ar l'introduction des applications conversationnelles et uelques solutions mises en oeuvre pour les résoudre. Nous résentons l'outil SAGE (Système d'Aide à la Gestion d'Ecrans) ont une "maquette" est actuellement opérationnelle sur un icro-ordinateur Micral 90-50.
otre étude porte sur les deux composantes principales du génie ogiciel : méthodes et outils.
ans une première étape, un cahier des charges concernant les roblèmes posés par les applications conversationnelles est dressé. De cette étude nous dégageons quatre points essentiels :

Un grande part du programme d'application est dédié au dialogue
mme-machine.

L'utilisateur final n'est pas suffisament inclus dans le
ocessus de développement du dialogue. Les principales décisions
nt prises par le programmeur.

Le programme d'application fait souvent appel à des tâches
pétitives que le programmeur est obligé de reécrire à chaque
uvelle application.

e cahier des charges nous permet de déduire les fonctionnalités
u système SAGE (système d'aide à la programmation d'applications
nversationnelles) :

séparer le dialogue du traitement en laissant tous les détails
u dialogue au système.

définir un éditeur d'écrans et de support de dialogue aussi
oche que possible de la conception de l'utilisateur.

réutilser au maximum des entités existantes.

faciliter l'utilisation des écrans dans un programme utilisa-
eur.
et enfin réaliser un outil portable pour une plus grande
iffusion.

autre part, pour une plus grande compréhension du système une
tude formelle à l'aide de types abstraits est entreprise.
ette étude sert d'une part de manuel d'administrateur (pour une
odification du système) et sert d'autre part de noyau de base
our les développements futurs du système.

## PLAN DE LECTURE.

Dans le premier chapitre, on trace un historique des fférents modes d'utilisation d'une installation informatique en s situant par rapport à l'évolution du matèriel, celle de utilisateur et enfin celle du dialogue Homme-machine.
ns ce même chapitre, on introduit le dialogue "pleine-page" : finition, intérêt et champs d'application.

deuxième chapitre est divisé en deux parties. Dans la emière partie, on essaie de dégager à partir d'un exemple estion simplifiée d'une bibliothèque) les caractèristiques des plications conversationnelles pleine-page et les outils et thodes nécessaires à leur mise en oeuvre. Dans la deuxième rtie, on présente, après avoir discuté quelques méthodes rmelles de description des applications interactives, le cadre éorique dans lequel s'inscrit le système SAGE.

troisième chapitre est une présentation du système SAGE. ns cette présentation nous abordons dans l'ordre : les jectifs du système, les options et concepts introduits (et ur justification) et enfin les fonctionnalités du système.

ns le quatrième chapitre, nous présentons une étude formelle s concepts du système SAGE à l'aide des types abstraits écifiés algèbriquement.
première partie de ce chapitre présente quelques généralités r les spécifications formelles et leur utilisation comme outil conception.

ns le cinquième chapitre on décrit l'architecture générale du stème validant les spécifications algèbriques présentées dans chapitre précédent ainsi que quelques caractéristiques nérales du système SAGE: portabilité, performance, évolutivité.

ns le sixième chapitre nous comparons SAGE à d'autres systèmes aide à la description d'applications conversationnelles, après oir établi, et discuté, un plan d'étude de ces systèmes.

fin le dernier chapitre conclut cette étude par une série de marques concernant l'apport de SAGE au développement des plications conversationnelles et quelques idées sur les tensions futurs.

# CHAP. I INTRODUCTION.

## .1. HISTORIQUE : LE TRAITEMENT PAR LOTS.

Le traitement par lots est par essence séquentiel. Il mposait une distinction nette entre les phases d'acquisition, de raitement et celle d'édition. Ceci est dû principalement au ype de matériels utilisés (caractérisés par des propriétés de équentialité ex : cartes, bandes magnetiques, listings 'imprimantes.....). Ces contraintes ont influencé de manière mportante tant le mode de travail du programmeur (cf. éveloppement des méthodes de programmation structurées) que la ise en oeuvre des applications informatiques. En effet, 'utilisateur de ce genre d'application est loin de la machine. l n'est pas maître des opérations qui se déroulent (saisie 'information, contrôle par programme, liste des erreurs et dition des résultats) et peut devenir un exécutant non motivé t finalement peu responsable. Les erreurs sont en effet orrigées plusieurs jours après leur introduction et l'édition es résultats est vue comme un phénomène déconnecté de 'acquisition de données.

## .2. LE DIALOGUE DANS UNE APPLICATION CONVERSATIONNELLE.

### .2.1. INTRODUCTION DES APPLICATIONS CONVERSATIONNELLES.

Avec l'arrivée des micro-ordinateurs et la généralisation u système à "temps partagé", les applications informatiques ont onnu un développement sensible. Au lieu de décomposer le éroulement de l'application en phases d'aquisition et raitement, le système conversationnel permet à l'utilisateur de ialoguer avec la machine, dialogue qui lui permet de corriger lus facilement certaines erreurs et de choisir l'information à ntroduire ce qui le rend plus motivé et donc plus responsable.

Mais ce développement, dont personne ne conteste l'intérêt rté à l'utilisation d'une installation informatique, ssite des travaux encore plus importants et une étroite aboration entre deux agents, tous les deux indispensables : ilisateur et le programmeur.

## 2. LE DIALOGUE : VUE DE L'UTILISATEUR.

Dans la conception d'un système conversationnel, tilisateur constitue une composante importante. Il est de s en plus admis que le succès d'un système interactif dépend grande partie de la qualité de l'interaction entre le système l'utilisateur.

Dans la prise en compte de l'utilisateur, on distingue trois mposantes:

### ature de l'utilisateur :

- "expert" ou novice. En effet, la forme du dialogue n'est du tout la même qu'il s'agisse d'un utilisateur "expert" ou ice. Dans le deuxième cas, l'utilisateur nécessite beaucoup s d'assistance, un langage de commande plus libre et facile à liser et à apprendre. Par contre, l'utilisateur "expert" rchera plus le moyen simple et rapide pour accèder aux ctions du système.

- Il faut aussi distinguer dans cette classification tilisateur occasionnel de l'utilisateur spécialisé, l'utilisa- ur ayant une connaissance en programmation....etc.

### Composante conceptuelle :

Tout utilisateur d'un système interactif se construit un dèle du sytème plus ou moins proche du modèle initial (du ncepteur) [M&V,82]. Le meilleur exemple pour illustrer cette ée est le système d'édition de textes. En, effet chaque ilisateur, suivant le nombre de commandes qu'il connait et la itrise plus ou moins importante du système, se construit un dèle du fonctionnement de l'éditeur.

Il s'agit dans ce cas pour le concepteur de rapprocher ces ux modèles en offrant à l'utilisateur une vision unique des bjets manipulés et en respectant le principe d'affichage : "ce 'on voit est ce qu'on obtient".

Composante matérielle.

Il s'agit de définir le poste de travail. Dans le domaine nous concerne ce poste se compose :

-d'un organe d'entrée : un clavier qui permet d'entrer des mmandes (à l'aide d'un langage de commandes) ou par l'utilisa-on de touches de fonction et un dispositif de sélection urseur, souris,....).

-un organe de sortie : un écran qui peut être :

* simple
* de type "bitmap"
* couleurs, graphique...
* .....

## 2.3.LE DIALOGUE : VUE DU PROGRAMMEUR.

L'introduction d'applications conversationnelles nécessite la part du programmeur des travaux encore plus importants. une part, elle nécessite la mise au point d'un langage de scription du dialogue. D'autre part, elle nécessite une dification des systèmes informatiques logiciels et matériels: conception du dialogue ne sera pas du tout la même si l'écran t géré en mode ligne ou en mode page. Il faut donc de uveaux outils.

Elle nécessite, enfin, le développement de nouvelles thodes, les méthodes de programmation structurées classiques ant indaptées pour ce genre de problème.

# 3. PROBLEMES POSES PAR LES APPLICATIONS CONVERSATIONNELLES.

## 3.1. LIMITES DES LANGAGES CLASSIQUES.

Les langages actuels n'ont pas été conçus dans le but de crire des algorithmes conversationnels :

-ils manquent d'outils linguistiques permettant de décrire acilement un écran. Les concepts qui s'en approchent sont ceux formats (cf. FORMAT en FORTRAN et PICTURE en COBOL,...) qui nt essentiellement consacrés aux éditions et saisies ligne-à-igne et ignorent les écrans "pleine-page".

-Ils ne permettent pas facilement d'exprimer les nchaînements de dialogue. Plus précisement, ils nécessitent de élanger traitements et dialogue, les seules enchaînements ossibles s'expriment par des structures de contrôle de bas

veau.

## 3.2. ABSCENCE DE METHODES.

L'abscence de méthodes pour développer des applications teractives a été signalée par plusieurs chercheurs [LAF,77], T,71]. Ainsi dans [NEW,71] on lit :

"Malgré l'intérêt de plus en plus croissant pour les tèmes interactifs, peu de travaux ont été faits sur les hniques et langages de description de dialogue homme-machine. plupart des applications interactives sont écrites dans un gage classique auquel des extensions ont été apportées pour prise en compte de l'interactivité".

D'autre part les méthodes classiques dérivées du courant de "programmation structurée" ne permettent pas facilement et turellemnt de concevoir un dialogue interactif [F&S,83]. C'est particulier le cas lorsque le dialogue s'éxprime sous forme arbre de menus et qu'on veut laisser à l'utilisateur la ssibilté de remonter plus d'un niveau dans cet arbre. Il faut ors prévoir des mécanismes pour décrire ces transitions régulières comme de boucles "exit" ou les "exception" ce qui existe pas dans les constructions de base de la programmation ucturée.

## 3.3. CARACTERISTISQUES DES PROGRAMMES CONVERSATIONNELS.

Nous présentons ci-suit quelques critères qui interviennent ns l'utilisation et la conception d'une application nversationnelle. Certains sont communs à toute application formatique (adaptabilité, traitement des erreurs, évolutivité) is prennent encore plus d'importance dans le cas des plications conversationnelles. Ces caractéristiques sont de ux types :

Caractéristiques "directes": ce sont les caractéristiques liés l'application relativement aux services attendus par l'utilisa- ur final. Elles comprennent :

a. Facilité d'utilisation : c'est la caractéristique qui téresse certainement le plus l'utilisateur final. Entrent dans définition de cette caractéristique les points suivants :

- Clarté du mode d'utilisation du système.
- Bon niveau de détail de cette description en fonction de la catégorie de l'utilisateur (cf. § I.2.2).
- Présentation claire des informations (possibilité d'accès à tout moment au mode d'emploi, documentation, fonctions d'aide ...).

b. Facilité d'apprentissage : Il s'agit dans ce cas aussi de s'adapter aux catégories des utilisateurs du système.

c. Traitement des erreurs : Il est important que chaque erreur détectée donne lieu à un commentaire clair qui permet à l'utilisateur de se retrouver facilement dans un état connu. Pour ce faire il est nécessaire de construire une hiérarchie des erreurs (syntaxiques, cohérence au niveau des zones d'un écran,..).

2) Caractéristiques indirectes : Elles sont liées à la conception de l'application (donc perçues par le programmeur) et qui peuvent avoir une influence sur les caractéristiques précédentes :

a. Adaptabilité : Il est nécessaire que le système puisse s'adapter à son environnement (besoin et catégories des utilisateurs).

b. Extensibilité : Le programme d'application peut faire l'objet de modifications plus ou moins importantes suite aux sollicitations de l'utilisateur. Une meilleure solution pour prendre en compte ces contraintes, est de donner à l'utilisateur la possibilité de définir lui même les extensions souhaitées.

## I.4. EVOLUTION DU DIALOGUE.

Cette évolution a été permise par les moyens technologiques mais répond aussi à un besoin des utilisateurs qui souhaitent disposer plus facilement de leurs informations et d'établir avec la machine un dialogue plus naturel.

### I.4.1. LE DIALOGUE LIGNE-A-LIGNE.

Cette forme de dialogue caractérise les premiers systèmes à clavier-imprimante ("télétypes"), aujourd'hui devenus largement obsolètes, et certaines consoles à écrans cathodique. C'est aussi sous cette forme que sont apparu les premiers éditeurs de textes encore utilisés aujourd'hui dans certains systèmes. L'utilisateur converse avec le système, dans ce mode, par une suite de questions/réponses. La correction d'une erreur nécessite généralement la reintroduction de toute la chaine. Ce mode de dialogue convient très bien pour une saisie simple (entrée de paramètres, commandes...) mais devient très vite fastidieux et inadapté pour des saisies complexes (traitement de texte, réservation de places,....).

## I.4.2. LE DIALOGUE PLEINE-PAGE.

Une caractèristique importante des terminaux modernes est qu'ils peuvent être traités comme des pages à deux dimensions et non pas seulement comme une suite de lignes. Le matériel offre donc la possiblité à l'utilisateur de préparer ses données en utilisant un écran complet. Il communique avec l'ordinateur en envoyant et recevant des messages qui ne remplissent non pas seulement une ligne mais plus généralement un "pavé rectangulaire" c'est ce qu'on appelle le dialogue "plein-écran" ou "pleine-page".

### I.4.2.1. CHAMPS D'APPLICATION.

Parmi les applications utilisant ce mode de dialogue citons:

-les éditeurs de textes "pleine-page" tels qu'ils existent dans plusieurs systèmes (EMACS, TED, SPF, ..). Le grand avantage de ces éditeurs est qu'il peuvent traiter un document comme une suite de pages non nécessairement limitées. L'utilisateur ne voit à un instant donné qu'une partie du document à travers l'écran, mais a la possibilité de se déplacer dans les quatre coins du document. Le succés obtenu par ces éditeurs auprès des utilisateurs rend extrèmement désagrèable le retour aux éditeurs ligne-à-ligne.

-La mise au point de logiciels ou l'utilisation de programmes par des non spécialistes sous le contrôle de menus successifs.

Parmi les applications de dernière technique citons l'exemple de l'informatique de gestion (système transactionnel) et l'enseignement assisté par ordinateur où l'utilisateur est souvent un débutant qui nécessite une assistance importante et un mode de dialogue et d'affichage le plus lisible possible.

### I.4.2.2. INTERET DU DIALOGUE PLEINE-PAGE.

L'interêt du dialogue pleine-page provient des quatre caractéristiques suivantes :

- l'utilisateur dispose à tout instant d'une vue d'ensemble s'étendant sur une page complète et non pas seulement sur une ligne.

- l'emploi de la page comme unité de communication avec l'ordinateur, qui permet à l'utilisateur de remplir les zones en suivant le parcours qui le satisfait, de corriger les éventuelles erreurs avant de transmettre une commande.

- la possibilté pour le système d'afficher des valeurs par

aut que l'utilisateur modifiera seulement si c'est nécessaire.
-personnaliser le dialogue en gardant pour chaque lisateur un profil lui évitant de répondre à des questions ntiques ou qui n'ont pas de sens dans son cas particulier.

A titre d'exemple, on présente dans la fig.I.1 un écran struit à l'aide du système SAGE où l'on voit apparaître lques unes de ces caractéristiques.

```
                COMPILATION PASCAL

     Nom fichier :  NUL
     Type        :  PAS
     Listing compilation : IMP

     options : .
               .
               .
               .

                          erreur : fichier inéxistant
                          xxxxxxxxxxxxxxxxxxxxxxxxxxx
```

fig. I.1

## 4.3. OUTILS ET METHODES.

La mise en oeuvre du dialogue pleine-page nécessite le éveloppement de nouveaux outils logiciels et des remises en uestion serieuses des méthodes classiques de programmation tructurées.

D'une part, il nécessite la mise au point d'un langage de escription et d'enchainement d'écrans permettant de caractériser n masque d'écran de la façon la plus souple possible. D'autre art, il nécessite le développement d'un système de mise en euvre de cette description et intégrant une méthode de onstruction.

Traditionnellement, les applications interactives (interface ideo) sont écrites dans un langage classique auquel certaines extensions ont été apportées (ou sous forme d'utilitaires) pour réaliser des fonctions d'affichage sophistiquées. Cette réalisation directe a plusieurs inconvénients :

- Réalisation fastidieuse : il faut réecrire une grande part des utilitaires du système.
- Manque de généralité.

- Les modifications sont rendus difficiles puisqu'on alise des structures "cableés".

Par ailleurs, les outils proposés jusque là par les nstructeurs sont d'accès difficile (cf. SPF [IBM], VFORMS II], ...) et ne permettent pas de réaliser facilement des quettes prouvant à priori la spécification du logiciel application.

Certaines équipes de recherches ont développé des ogiciels de gestion d'écrans (cf. PIGME [SCH,81], GESCRAN ES,80], TTV [GRE,80] ). Ces travaux ont permis d'une part de velopper des outils généraux dont un des objectifs est la rtabilité et d'autre part de mieux cerner les problèmes et ractéristiques de ce type de logiciel. Les idées issues de s travaux, en particulier PIGME et GESCRAN ont beaucoup fluencé la réalisation du système SAGE.

## 5. LE DIALOGUE DANS LES ENVIRONNEMENTS DE PROGRAMMATION.

Un autre pas a été franchi ces toutes dernières années dans e domaine de l'interface homme-machine par le développement de ertains laboratoires de recherches de postes de travail phistiqués utilisant un matériel spécial (généralement des erminaux de type "bitmap" et un dispostif de sélection : la souris") et un système d'exploitation adéquat.

Parmi ces systèmes citons les nombreux outils construits utour de LISP [SAN,78] et la langage SMALL-TALK mis au point u centre de recherche de XEROX PALTO ALTO [BYT,82].

Ces environnements utilisent les techniques suivantes :

-le multifenêtrage qui permet à l'utilisateur de gérer au ieux son écran.

-L'abscence de mode qui permet à l'utilisateur de ravailler sur plusieurs activités simultanément.

-la technique de manipulation directe qui permet de désigner irectement un objet sur l'écran.

-enfin la mutireprésentation qui permet de visualiser un même objet sous deux aspects différents (par exemple texte et graphique).

# HAP.II GENERALITES SUR LES APPLICATIONS CONVERSATIONNELLES

## .1. INTRODUCTION. OBJECTIF DE L'ETUDE.

L'objectif de cette partie de l'étude est de dégager, à rtir d'un exemple, les caractéristiques et les outils cessaires à la mise en oeuvre des applications interactives. s caractéristiques seront étudieés principalement sous aspect dialogue et interface avec l'utilisateur. Mais nous rrons que cet aspect peut avoir d'importants effets sur la nception de l'application.

L'étude de ces caractèristiques nous mènera aussi à scuter de la structure des programmes de dialogue et donc des thodes permettant de programmer efficacement ces applications. s méthodes doivent satisfaire un besoin essentiel ressenti tuellement dans le développement d'environnements de programma- ion qui est celui de manipuler, indépendament des traitements l'application, le dialogue.

Nous verrons enfin comment ces reflexions seront ncrétisées par la réalisation d'outils logiciels.

## .2. ETUDE D'UN EXEMPLE.

### .2.1. L'EXEMPLE : GESTION SIMPLIFIEE D'UNE BIBLIOTHEQUE.

Cet exemple a été choisi pour introduire les applications teractives de part l'importance de l'interface avec utilisateur (en volume d'échange et de représentation informations) et aussi l'utilisation des bases de donneés qui présente un cas fréquent où la conception d'un interface omme-machine constitue une part importante du système [SHE,80], [LO,75].

Supposons qu'une bibliothèque décide d'automatiser certaines pérations en utilisant des terminaux conversationnels reliés à ordinateur.

emarque :

Dans cette partie de l'étude, nous nous limiterons à la escription externe de l'application en supposant en particulier u'une étude préalable et la recherche d'une solution nformatique justifiant l'utilisation de moyens conversationnels été déja faite [HER,82].

Parmi ces opérations on trouve :

- prise en compte de l'emprunt d'un ouvrage par un abonné.
- retour du livre par l'abonné.
- enregistrer, supprimer ou accèder aux caractéristiques d'un abonné.
- enregistrer, supprimer ou accèder aux caractéristiques d'un ouvrage.
- commander un livre chez un éditeur.
- enregistrer, supprimer ou accèder aux caractèristiques d'un éditeur.
- liste des états récapitulatifs :
  - * par abonné : liste des ouvrages empruntés
  - * par éditeur : liste des ouvrages commandés.
  - * liste des abonnés
  - * liste des ouvrages
  - * liste des éditeurs.

Remarque :

Cette application représente un exemple type où le problème de dialogue est important par rapport aux traitements (qui consistent dans ce cas en des mise-à-jour de bases ou fichiers

ationnels si l'on utilise des bases de donneés relationnelles)

Face à ce type de problème, il serait agréable de disposer outils qui permettent de décrire l'aspect externe de pplication (résultats et dialogue) et pouvant donner une idée le système avant sa réalisation complète. Ces outils seront utant plus intéressants et indispensables s'ils impliquent la ticipation active de l'utilisaeur final [MAR,73] S,84],[MEY,84]. En effet, c'est un rare cas où des outils ormatiques servent de support de dialogue entre informaticiens utilisateurs.

## .2.2. DESCRIPTION DU DIALOGUE.

### .2.2.1. L'ENCHAINEMENT DES ECRANS.

La première étape de conception du dialogue consiste à crire précisement l'enchaînement des écrans en fonction des ponses de l'utilisateur. Pour ce faire les graphes de ansitions, analogues à ceux utilisés en informatique de stion et compilation, apparaissent comme un bon moyen pour crire le déroulement du dialogue [FAU,83]. A chaque noeud du aphe est associé l'écran devant être affiché lorsque ce noeud t atteint. Les transitions représentent les réponses de utilisateur ou éventuellement les conditions pour passer au eud suivant. La fig. II.1 présente le graphe obtenu pour application bibliothèque :

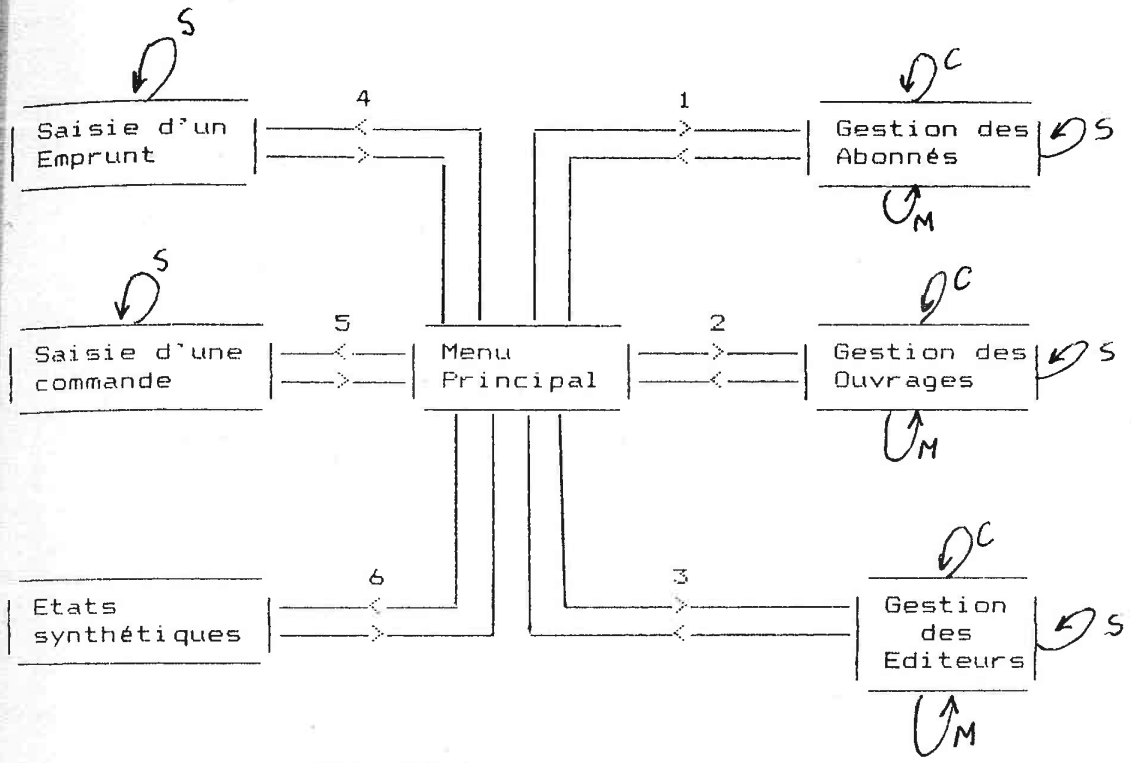



fig. II.1

égende :

C = création
S = suppression (ou suivante)
M = modification

I.2.2.2. STRUCTURE DES INFORMATIONS DU DIALOGUE.

Le corps du dialogue proprement dit consiste en la suite es écrans (voir fig. II.2..II.5)

```
                APPL. GESTION BIBLIOTHEQUE

                    MENU PRINCIPAL

        1 -->  Gestion des abonnés.
        2 -->  Gestion des ouvrages.
        3 -->  Gestion des éditeurs.
        4 -->  Saisie d'un emprunt .
        5 -->  Saisie d'une commande.
        6 -->  Etats synthétiques.
        7 -->  Sortie.

                                        Choix : .
```

fig. II.2.

```
                APPL. GESTION BIBLIOTHEQUE

                  GESTION DES ABONNES

        No. abonné : ......

        Nom : ........................     Prenom : ......

        Adresse :
                Rue : ................    No. : ...
                Ville : ..............    Code postal : .....

    C ==>Creation , S ==>Suppression , A ==>Accès , F ==>Fin

                                        Choix : .
```

fig. II.3

APPL. GESTION BIBLIOTHEQUE

SAISIE D'UN EMPRUNT

No.abonné : .....

Nom : ..... Prenom : .....

No.livre : ..... Auteur : .....
Titre : .....

Date emprunt : ../../.... Date retour : ../../....

Validation ==> RC , Annulation ==> Esc

fig. II.4

APPL. GESTION BIBLIOTHEQUE

ETATS SYNTHETIQUES

Listes des ouvrages empruntés par :

No.abonné : .....
Nom : ..... Prenom : .....

| Numero : | Auteur | Titre | Date emprunt | Date retour |
|---|---|---|---|---|
| ..... | ..... | ..... | ../../.... | ../../.... |
| ..... | ..... | ..... | ../../.... | ../../.... |
| ..... | ..... | ..... | ../../.... | ../../.... |

fig. II.5

On peut tout de suite remarquer que l'utilisateur souhaite rire plusieurs types d'écrans :

- des écrans de type menus (fig. II.2).
- des écrans de type saisie d'informations (fig.II.4).
- des écrans d'affichage d'informations (fig.II.4).
- des écrans d'aide (cette possibilité n'a pas été représentée dans l'exemple).

Et enfin, plus généralement une combinaison de ces fférents types (fig. II.3).

D'autre part dans chaque écran l'utilisateur spécifie des ructures d'affichage décrivant la représentation visuelle des jets qui vont servir de support de dialogue avec utilisateur final.

Ces structures se répartissent en plusieurs types :

des structures élémentaires.
exemple : No. abonné : .....

des structures plus complexes regroupant plusieurs informations élémentaires et qui peuvent être utilisées telles quelles dans plusieurs écrans.
ex : Adresse

des informations à structures tabulaires permettant de visualiser des informations répétitives.
ex : Liste des ouvrages empruntés.

En plus des outils de description du dialogue, il est utile e disposer d'un langage de spécification de contrôles. Ce angage doit permettre de specifier des contrôles du type format:

ex : supposons que le No.abonné soit codé sur 2 caractères lphabétique suivi de 3 caractères numériques. Le format orrespondant est AANNN
avec possibilité de cadrage (à droite ou à gauche) et de emplissage par défaut (avec des blancs ou zéros).

D'autre part le langage doit permettre de spécifier des ontrôles dynamiques plus complexes liés :

- soit une zone élémentaire :

ex : - zone obligatoire ou facultative.
- la valeur de la zone appartient à un intervalle donné.
- la valeur de la zone appartient à un ensemble de valeurs données.
- la valeur de la zone > , < ou = à une valeur donnée.

-soit à un groupement de zones. Dans ce cas la cification des contrôles peut s'exprimer par une liste de les de la forme :

P(zone) si <condition>

ou P est une propriété quelconque que doit vérifier la eur d'une zone (cf. ci-dessus) et <condition> une expression ique portant sur les valeurs d'une ou plusieurs autres zones.

ex :

No_abonné ∈ [1..1000]
Date_retour > Date_emprunt
Nbre_ouv =< 3 si Cat_abonné ='Etudiant'

Remarquons que le style déclaratif de spécification des trôles correspond bien à notre intention d'utiliser un mode erprétatif de description du dialogue. En particulier tilisateur ne s'occupe ni de l'ordre des propriétes ni de r "sémantique".

## 2.3. MODELE GENERAL D'ETUDE D'UNE APPLICATION CONVERSATIONNELLE "PLEINE-PAGE".

L'exemple précédent donne un cas type d'une application nversationnelle "pleine-page". L'étude d'une telle application sse par les étapes suivantes :

1. Faire l'inventaire des écrans de dialogue. Ces écrans répartissent en plusieurs types et utilisent des structures affichage déterminées par le programmeur et/ou l'utilisateur nal.

2. Définir l'enchaînement des écrans. Cet enchainement t tenir compte de tous les cas possibles en fonction des ponses de l'utilisateur final.

3. Enfin définir les traitements à faire après affichage et sie de chaque écran ; traitement qui consistent souvent en des se-à-jour de fichiers ou bases de données.

Les premiers outils mis à la disposition du programmeur nt fournis par les constructeurs. Très différents dans la ésentation, le mode d'utilisation et le materiel visé ; tous t pour objectif de faciliter le développement d'applications teractives en proposant au programmeur un langage de scription d'écrans (cf. chap.vi). Mais ces outils aussi

phistiqués qu'ils peuvent être ne sont pas suffisants pour ogrammer efficacement les applications conversationnelles leine-page". En effet une des caractèristiques principale de genre de logiciels (précisement celle qui caractérise le alogue interactif) est leur structure d'ensemble et évolutivité des programmes [CHT,82]. En effet le programme de alogue peut faire l'objet de modifications plus ou moins portantes suite aux sollicitations de l'utilisateur. Parmi s modifications qui peuvent intervenir dans le déroulement du ogramme, l'utilisateur final peut décider de disposer trement les informations qu'ils lui sont présentées. De la même açon il pourrait décider d'ajouter ou supprimer d'autres formations ce qui pose le problème de cohérence. Un autre emple est celui où l'utilisateur décide que l'enchainement des rans prenne une autre forme. Il en résulte que le programme it tenir compte de toutes ces contraintes et être très volutif et fortement paramétré [MEY,82],[WAS,84].

Dans la conception d'un système interactif, l'utilisateur inal avec sa diversité et ses contraintes, constitue une mposante importante dont il faut en tenir compte. Une des eproches généralemnt faite aux systèmes interactifs classiques st que le concepteur doit imaginer, tout le long de la rogrammation de son système, l'aspect qu'aura celui-ci face à 'utilisateur final.

Il faudrait donc en plus de ces outils logiciels développer ne méthode aidant à l'écriture des programmes de dialogue en enant compte de ces contraintes et qui permettrait par illeurs de construire une spécification externe de 'application donnant la possibilité d'établir des maquettes.

Cette discussion fait l'objet de la deuxième partie de ce hapitre.

## 3. SUR LA STRUCTURE DES PROGRAMMES DE DIALOGUE.

### 3.1. LIMITES DES LANGAGES ET METHODES CLASSIQUES.

En général les langages et techniques classiques de ogrammation des applications interactives ne permettent que spécification de procédures exécutées à la suite d'actions de tilisateur. Ces actions sont "détéctées" par des procédures éciales analogues aux procédures d'"exception" disponibles s certains langages de programmation tels que ADA,CLU ou /1, appelées dans ce cas "attention- handling".

La mise en oeuvre de ces procédures est généralement faite quatre façons :

-description tabulaire reliant actions et procédures.

-test conditionnels de la forme :
si <expression logique> alors <instruction>

-déclaration de procédures :

```
x: ON LIGHTPEN DETECTED GOTO LABEL 1
   IF KEY5 THEN CALL KPROG5
```

-ou enfin sous forme de diagrammes de transitions.

Ces approches ont plusieurs inconvénients :

-description procédurale : le programmeur pense en terme de élation entre résultats et actions, mais il est obligé de les anslater en une suite d'instructions.

-le programmeur est obligé de tenir compte de toutes les nditions possibles.

-l'exécution superflue de code puisque le programme doit ester toutes les conditions, conditions qui sont souvent dentiques d'une interaction à une autre.

D'autre part ces techniques ne permettent de définir aucune pécification externe de l'application (cf. § II.2.4). Ceci a nduit naturellement à l'élaboration de méthodes plus formelles ncluant les notions d'abstraction et de programmation dirigée ar les données. Ce sont ces méthodes qui vont être discutées ns les paragraphes suivants.

## .3.2. DEUX METHODES DE SPECIFICATION DU DIALOGUE HOMME-MACHINE.

### .3.2.1. LA NOTATION BNF.

La notation BNF introduite par Backus est largement utilisée ur décrire la syntaxe des langages. Plusieurs auteurs ont ilisé cette notation pour spécifier le dialogue Homme-machine lui apportant certaines modifications [JAC,83]. Ces difications consistent à associer des actions aux règles de ammaire, actions qui sont invoqueés à chaque application de s règles. On va illustrer cette méthode sur l'exemple de nnexion à un ordinateur : la procédure Login.

Cette commande demande d'abord à l'utilisateur de rentrer n nom. Si le nom rentré est correct l'utilisateur doit entrer suite un mot de passe, sinon la même question est posée squ'à ce qu'il tape un nom correct. Le mot de passe est suite controlé, s'il est correct la connexion est établie non il doit le rentrer. Si après (Limite) tentaives, le mot de asse est toujours incorrect l'utilisateur est obligé de ecommencer toute la procédure.

```
gin ::=LOGIN <Ident> <Mauv.mp>* <Bon mp>

dent ::=[rep: "entrer identité "] <Mauv.ident.>* <Bon. ident.>

auv.ident ::= cond: non_exist($ident)
              [rep: "identité inexistante _ reesayer " ]

on.ident ::= cond : exist($ident)
             [rep : "entrer mot de passe " ]

auv.mdp ::= cond: non_exist($mot_de_passe) et $compt < Limite
            [rep:"mot de passe érroné _ resssayer "]
            act : $compt=$compt+1

on.mdp ::= cond: exist($mot_de_passe)
           [rep:"connexion établie "]
```

La clause cond indique la ou les conditions nécessaires à application de cette règle de production. Act signifie exécution d'une action qui peut être l'appel d'une procédure ystème.

Une spécification basée sur la notation BNF peut être tilisée comme entrée à un compilateur où les actions associées ux règles sont décrites sous forme de procédures exécutables. armi les applications de ctte technique citons la

nstruction de générateur de prototypes de dialogue [KAY,82], RI,84].

### I.3.2.2. LES DIAGRAMMES DE TRANSITIONS.

Ces diagrammes se rapprochent des automates d'états finis ilisés en compilation et en informatique de gestion. lusieurs auteurs ont utilisé les graphes de transitions pour crire le dialogue Homme-machine [WAS,84]. Ces travaux différent ns la complexite du système specifié. Dans cette approche, le ialogue est découpé en un ensemble d'états . Les graphes de ansitions doivent être modifiés d'une facon analogue aux otations BNF. A chaque transition sera associée une action, aquelle sera exécutée si cette transition a lieu.
a fig.II.3. décrit le graphe de la commande Login vue récédement ainsi que la liste complète des transitions et états u dialogue :

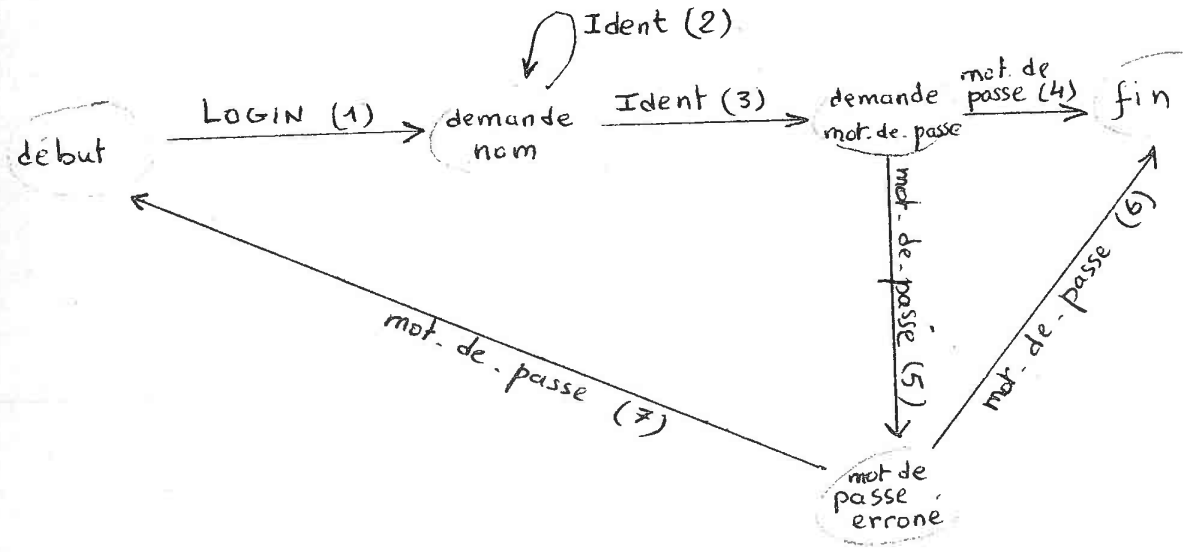



fig. II.6

lotations :

Les états sont représentés par des cercles. Les ansitions sont représentées par des arcs directs numérotés. and une transition a lieu on indique la réponse du système et s actions exécutées.

```
(1) : rep: "entrer identité "

(2) : cond: non_exist($ident)
      rep : "mauvaise identité _ retaper "

(3) : cond: exist($ident)

(4) : cond: exist($mot_de_passe)
      rep : "connexion établie"

(5) : cond: non_exist($mot_de_passe)
      rep : "mot de passe erroné _ ressayer "
      act:  $compt=$compt+1

(5'): cond:non_exist($mot_de_passe) et $compt < Limite
      rep :"mot de passe eronné _ ressayer "
      act : $compt=$compt+1

(6) : cond: exist($mot_de_passe)
      rep : "connexion établie"

(7) : cond: non_exist($mot_de_passe) et $compt = Limite
     rep : "recommencer la connexion "
```

Remarquons que bien que ces deux méthodes de spécification ont formellement équivalentes, la différence entre elles tient rincipalement à la compréhension. Alors que les méthodes basées ur les diagrammes de transitions contiennent explicitement la otion d'état ; cette notion est implicite dans les notations BNF.

### I.3.3.2. DISCUSSION.

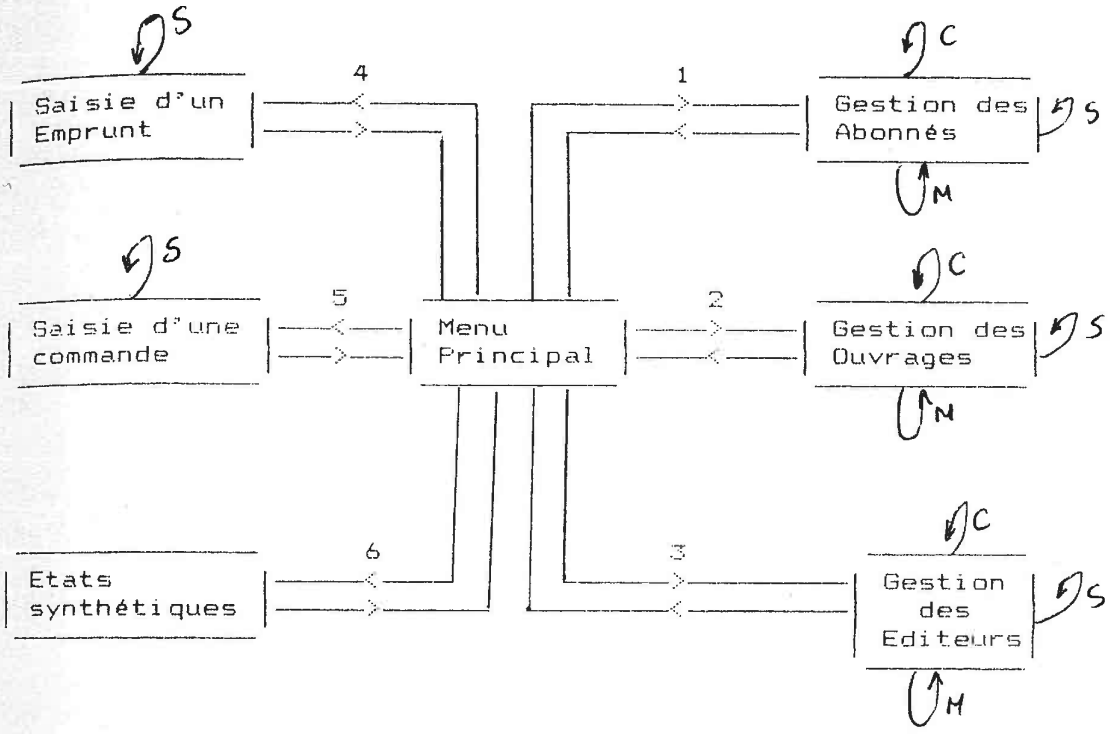
L'étude de ces méthodes appelle au moins deux critiques :

- la première concerne la formalisation du dialogue omme-machine. L'une et l'autre de ces méthodes essayent de ormaliser ce dialogue en imposant des règles (explicites dans e cas de la notation BNF) telle que l'on aurait fait pour un angage de programmation par exemple [BRN,82]. Or, il nous semble u'il est dangereux de fonder la conception d'un interface omme-machine par analogie avec celle d'un autre outil formel e communication avec la machine tel qu'un programme. Face à un ystème intéractif, le comportement de l'utilisateur final peut tre d'une complexité arbitraire et n'a aucune raison de suivre es règles initialement prévues. Comme exemple de contre-éactions de celui-ci citons : les court-circuits, parce ue l'utilisateur souhaite passer directement d'un état à un utre sans passer par le chemin initialement prévu ; les uspensions momentanées du dialogue par les touches "exit" ou es commandes d'"aide". C'est le système qui doit être au ervice de l'utilisateur et non le contraire.

- La deuxième remarque concerne la forme du dialogue. Les principaux travaux utilisant ces méthodes se sont limités à un dialogue du type ligne-à-ligne sans s'occuper du fait qu'il existe d'autres formes de dialogue. Comme nous le verrons dans les sections suivantes, le passage à un dialogue "pleine-page" introduit de nouveaux problèmes qui exigent des réflexions serieuses sur les outils et méthodes de programmation.

## II.3.3. CARACTERISTIQUES DES APPLICATIONS CONVERSATIONNELLES "PLEINE-PAGE".

La structure d'une application conversationnelle pleine-page peut être souvent modélisé par un graphe de transitions. L'exécution du programme se traduit par le parcours d'un chemin dans ce graphe. Il en est ainsi de l'application bibliothèque :




A chaque pas de l'exécution, correspondant à un noeud du graphe le programme affiche à l'utilisateur un écran ; lequel remplit certaines (ou toutes les zones) et actionne une touche de fonction correspondant à son choix. Ensuite, le contrôle retourne au programme pour vérifier la validité des réponses et éventuellement les enregistrer.

La réalisation directe de ce schéma dans un langage classique (typiquement un langage procédural) se traduirait par l'écriture d'un certain nombre de procédures (en fait exactement une par état) du type :

```
proc. etat
debut
   afficher(ecran)
   repeter
      lire(reponse,choix)
      controle(reponse)
      si erreur alors message
   jusqu'à pas erreur
   enregistrer(reponse)
   cas choix ou
    c1 : etat_1
    c2 : etat_2
    ..........
    cn : etat_n
finproc.
```

Ce style de programmation conduirait naturellement à des programmes intexricables de type "plat de spaghetti". Ceci est du fait, déja signalé (cf.§ II.3.2), que le graphe de transitions peut être d'une structure arbitraire (en effet il faut prévoir des détours temporaires : touches "aide" exprimée dans l'exemple par le caractère ?). L'utilisateur est souvent soummis à un enchaînement rigide, il souhaiterait passer directement d'un noeud à un autre sans qu'il ait besoin de traverser un chemin complet.

## II.3.4. APPROCHE PRECONISEE: APPLICATION DIRIGEE PAR LES DONNEEES

La difficulté dans la méthode procédurale provient du fait qu'il n'y a pas de séparation entre entrée/sorties de données et traitements. Les fonctions telles que contrôle de validité et édition font souvent partie du traitement.

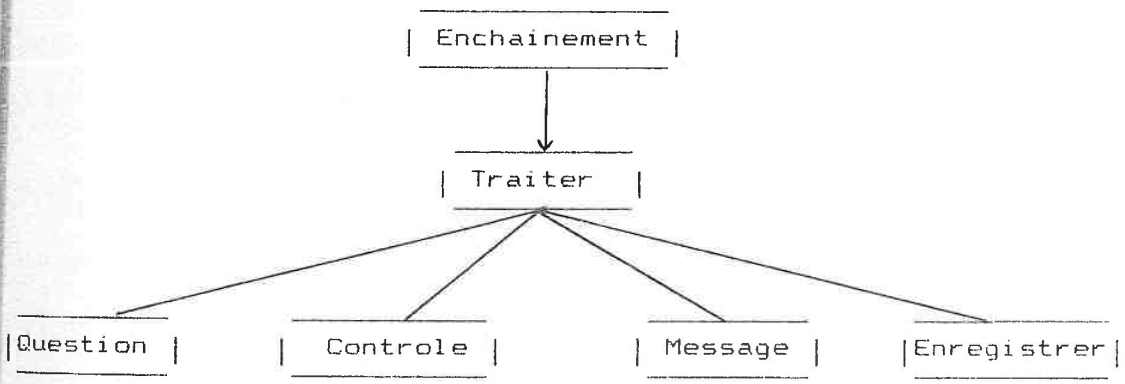
Une meilleure approche, à notre avis, est celle qui consiste à décrire indépendament dialogue et traitements. Ces derniers seront programmés selon les méthodes classiques. Pour décrire le dialogue, l'utilisation de la commande par table qui se rapproche des tables de décision utilisées en informatique de gestion et les automates bien connus en commande de processus et en compilation, est d'une grande aide pour le but poursuivi. Elle considère le système conversationnel comme un système à états dont l'évolution consiste en une série de transitions. Un

tat est vu comme une unité de dialogue indépendante mais avec us les renseignements nécessaires à son activation. A tout tat sera donc associés un certain nombre de caractèristiques :

attributs de l'état : numéro associé, écran affiché lorsque le système se trouve dans cet état.
opérations pouvant être demandées lorsque cet état est atteint:

* Question
* Contrôle
* Message
* Enregistrement

Pour décrire le programme conversationnel, on utilisera six nités de programmes sur trois niveaux hiérarchiques :




Le programme Enchainement définit seulement le cheminement dans le graphe, il ne connait rien des écrans propres à une application (et doit donc être utilisé identique pour des applications différentes).

```
proc. Enchainement
  var e : etat ; t : touche
  e := etat_initial
  repeter
     Traiter(e,t)
     e:=Transition(e,t)
  jusqu'à e=etat_final
fproc.
```

"Transition" est la fonction qui décrit le graphe des ats. Transition(e,t) est l'état obtenu lorsqu'on quitte état e par la voie t (t désigne dans notre cas le numéro de ouche de fonction). En suivant le principe précédent on eprésentera Transition par un tableau à deux dimensions (matrice ransitions). Le programme est alors insensible aux politiques e cheminement. En ce sens on définira un graphe programmable lutôt que de le cabler une fois pour toute.

"Traiter" décrit les traitement à faire dans un état donné : oser une question, lire la réponse de l'utilisateur, contrôler ette réponse et exécuter les actions nécessaires en fonction de a réponse donnée.

```
roc. Traiter(e,t)
  var r : reponse ; correct : boolean
  repeter
    t,r:=Question(e)
    correct:=Controle(r,e)
    si non correct alors Message(r,e)
  jusqu'à correct
  Enregistrer(r,e)
fproc.
```

Question, Controle, Message, Enregistrer sont ceux,spécifiques à chaque état. Question affiche l'écran associé à un état et lit la réponse de l'utilisateur.

```
proc. Question(e)
  Afficher(ecran_associe(e))
  Lire(reponse)
fproc.
```

Contrôle(r,e) vérifie la réponse r pour cet état. Message(r,e) affiche si nécessaire un méssage d'erreur et enfin Enregistrer(r,e) effectue les actions résultant de la prise en compte de la réponse r pour l'état e.

Pour programmer en pratique ce schéma en tenant compte des ux caractèristiques de ce genre de logiciel, i.e évolutivité et utilisabilté, il est nécessaire de chercher du côté des uvelles méthodes et techniques de programmation qui sont lles fondées sur les notions de types abstraits et ogrammation orientée objets. En effet les méthodes structurées assiques sont inadaptées pour ce genre de problème (cf. scussion dans [F&S,83],[MEY,82] et [STE,82]).

En suivant cette approche, la structure du programme sera sée autour de deux objets (types abstraits) principaux :

- Le type Application, qui décrit la forme générale du alogue avec ses caractéristiques :

  * Etat initial du dialogue
  * Etat final
  * la procédure Enchainement décrite précédement.

- Le type Etat avec les opérations le caractérisant vues us haut.

La réalisation de ce schéma dans un langage d'implémenta- on de types abstraits tel que SIMULA (cf. § IV.1.6) consiste la description de deux classes :

```
lass Application;
egin
 comment attributs;
   ref(Etat) etat_initial,etat_final;
   ref(Etat) array transistion(1:NE,1:NT);integer NE,NT;
 comment NE : nombre d'états ; NT : nombre de touches;
 comment opérations;
   procedure Enchaînement;
   begin
     integer eti;
     ref(Etat) etat_courant;
     etat_cournat:=etat_initial;
     while not(etat_courant=etat_final)
     begin
       etat_courant.Traiter(eti);
       etat_courant:=transition(etat_courant,eti)
     end;
   end; enchainement
 comment actions;
 comment initialisation de transition
end; Application
```

```
ass Etat;
virtual
   ref(reponse) procedure Question;
   boolean procedure Controle
   procedure Message;
   procedure Enregistrer;
egin
comment attributs;
   integer Noe; numero de l'etat;
   ref(Ecran) Ecr_asoc; ecarn associé;
comment operations;
   procedure Traiter(Choix);integer Choix;
   begin
     boolean correct;
     correct:=false;
     while not correct
     begin
       ref(Reponse) r ;
       r:=Question(Choix);
       correct:=Controle(r)
       if not correct then Message(r)
     end;
     Enregistrer(r)
   end Traiter;
end Etat;
```

Le grand avantage de cette approche est qu'elle permet une construction véritablement modulaire. La partie indépendante de l'application est programmée séparément. Pour une application donnée, chaque état est ensuite décrit de façon entièrement autonome sans référence à l'enchaînnment du dialogue, mais avec toutes les caractéristiques de l'état sans exception : écran à

fficher, réponses à lire, contrôle de validité, message d'erreur t enregistrement des réponses validées.

D'autre part cette mèthode conduit naturellement au développement d'un système d'aide à la programmation des applications conversationnelles dont c'est l'objectif du système SAGE présenté dans cette thèse.

Concrètement ce système d'aide facilitera le développement d'applications intéractives en fournissant :

- Un outil (éditeur) permettant la construction et l'archivage des écrans .

- Des utilitaires regroupant toutes les opérations concernant l'utilisation d'un écran :

  * affichage
  * Saisie
  * Contrôle
  * Gestion des messages

- Un système de gestion des applications permettant l'évolutivité et la réutilisabilité des programmes.

## II.3.5. INTRODUCTION AU SYSTEME SAGE.

Le projet SAGE, "Système d'Aide à la Gestion d'Ecrans", s'insère dans le cadre du travail d'une équipe de recherche du CRIN "équipe programmation" qui vise à réaliser des systèmes d'aide à la spécification et construction de programmes. Le but poursuivi dans cet axe de recherche est d'améliorer le travail du programmeur en lui proposant des méthodes, des langages et des outils d'aide à la programmation.

L'équipe s'appuie, pour réaliser ses projets, sur les expériences acquises il y a quelques années dans les domaines suivants :

- spécification de problèmes et types abstraits.
- construction de programmes.
- analyse des systèmes conversationnels.

D'autre part la plupart des projets de l'équipe, dont SAGE, trouve leur origine dans un cas concret.Le mode de fonctionnement se présente ainsi : partant d'une étude expérimentale du problème, on développe un cadre formel et ensuite on propose des méthodes et outils logiciels adaptés.
Cette étude se solde généralement par la réalisation d'une maquette expérimentale du système permettant de valider les théories et idées préalablement proposées.

# CHAP-III LE SYSTEME SAGE (PRESENTATION GENERALE).

## III.1. INTRODUCTION.

Les principales conclusions de l'étude du chapitre précédent peuvent s'énoncer ainsi :

- une grande partie du programme d'application conversationnelle consiste à décrire l'interface avec l'utilisateur. Plus précisement, une enquête a montré que la part de logiciel dédié à la relation homme-machine représente jusqu'à 60% de la masse totale du code [SUT,78].

- La plupart des concepts de description de dialogue se retrouvent d'une application à une autre. Le programmeur passe souvent son temps à reécrire des parties de dialogue semblables (cf. partie contrôle).

- Inadaptation des langages de programmation et des méthodes classiques (méthodes structurées) fondées sur le concept de procédures. Les langages de programmation manquent d'outils linguistiques pour décrire facilement des masques d'écrans.

## III.2. OBJECTIFS DU SYSTEME SAGE.

L'objectif du système SAGE est de fournir un outil portable intégré et une méthode de spécification et de mise en oeuvre des applications conversationelles. Par outil integré on entend les fonctionnalités suivantes :

### III.2.1. Spécification et construction :

Il s'agit à partir d'une spécification d'une application conversationnelle de construire les supports de dialogue. Dans cette phase deux agents sont concernés : le programmeur et l'utilisateur final. En effet, l'utilisateur final intervient directement pour définir les extensions adaptées et les divers composants de l'application à savoir la forme du dialogue. En plus des supports de dialogue (Article, Scadre, Tableau, Menu et Cadre), l'utilisateur spécifie aussi les contrôles qu'il souhaite exécuter au moment de la saisie de données.

### III.2.2. CAO d'écrans-édition.

Pour manipuler les différents composants, l'utilisateur dispose d'un éditeur syntaxique qui lui permet d'effectuer interactivement toutes les opérations qu'il souhaite faire sur une description d'écrans. On peut alors parler d'une véritable construction assistée d'écrans.

### III.2.3. Archivage et contrôle.

Pour une utilisation efficace du système, on doit fournir à l'utilisateur la possibilité d'archiver une description d'écran pour une modification ultérieure. L'opération d'archivage s'étend aussi à des parties d'écrans que nous appelons Articles et Scadre. A chaque niveau de description l'utilisateur pourra créer, supprimer, modifier, dupliquer ou connaitre la liste des composants présents.

### III.2.4. Utilisation.

il s'agit par un mécanisme d'interprétation d'utiliser les écrans crées soit lors d'une phase de mise à point ou lors d'une exploitation réelle à la suite d'une sollicitation du programme utilisateur.

### Méthode :

Les meilleurs outils qu'on puisse fournir à l'utilisateur ne sont pas suffisants pour programmer efficacement les applications conversationnelles. Aussi faut-t-il ajouter à la réalisation de ces outils logiciels, le développement d'une méthode de programmation (ou un système d'aide à la

programmation). La méthode utilisée dans SAGE est celle d'application dirigée par les données exposée au chap.II § 2.4.

## III.3. CHOIX ET CONCEPTS.

### III.3.1 PRINCIPAUX CHOIX FAITS.

Pour atteindre ces objectifs plusieurs options ont été prises :

a- Séparer le processus de dialogue de l'application proprement dite afin de décharger le programmeur des détails fastidieux tels que (gestion de l'écran, contrôles répétitifs ..) tout en gardant la logique du programme. Cette option permet aussi d'atteindre une indépendance de l'application vis à vis du style de dialogue. La forme du dialogue peut changer sans que le corps de l'application en soit affecté.

b- Réutiliser au maximum les entités déja définies évitant à l'utilisateur de respécifier des descriptions existantes. Cette option permet aussi d'éliminer les incohérences dans le cas où plusieurs applications utilisent des entités communes. A l'aide d'une bibliothèque d'applications, l'utilisateur pourra même récupérer des parties d'applications.

c- Reporter les contrôles au niveau de la saisie évitant ainsi de solliciter l'application qui pourrait être gérer sur un autre ordinateur . Cette option a aussi l'avantage de pouvoir stocker des spécifications de contrôles.

d- Pour réaliser un système portable, il est plus judicieux de décrire les caractéristiques du terminal dans un fichier au lieu de les "câbler" une fois pour toute. D'autre part, la diversité des terminaux nous oblige à ne considérer que les caractéristiques de base (adressage du curseur, fonctions d'attributs visuels), les autres fonctions (protection, "mémoire",...) sont programmées sous forme d'utilitaires.

### III.3.2. CONCEPTS ET MODELES INTRODUITS.

La définition du système nous amène à introduire un modèle et un certain nombre de concepts dont on donne ci-dessous les définitions. La justification de ces concepts fait l'objet de la section suivante.

### III.3.2.1. LE CONCEPT D'APPLICATION.

Une application conversationelle consiste, pour la partie dialogue, en un ensemble d'écrans organisé sous forme d'un graphe orienté (cf. chap.II § II.3.3).
Une instance d'une application consiste à décrire la relation entre écrans qu'elle utilise (enchaînement) et les traitements sous-jacents.

### III.3.2.2. LE CONCEPT D'ECRAN.

L'écran (ou Cadre) est l'unité de dialogue. C'est un arrangement géographique de sous-écrans. Un sous-écran est soit un Menu, un Tableau ou plus généralement un Sous-Cadre ou un Article.

Un Menu consiste en un ensemble d'options (zones protégées ou parties fixes dans la terminologie de SAGE) et une rubrique (ou Article) éventuellement vide utilisé pour prendre en compte le choix de l'utilisateur. La sémantique d'un menu consiste simplement à s'assurer que la réponse choisie fait partie des options du menu.

Un Tableau d'Articles consiste en une collection d'Articles tous de même type. La description d'un tableau doit préciser l'Article de base, le nombre d'occurences d'apparition de cet Article et l'alignement souhaité (Vertical ou Horizontal).

Un Sous-cadre est un arrangement quelconque d'Articles.

L'Article est l'unité de base à partir duquel sont construits les autres composants. Un Article est formé d'un champ fixe (protégé) éventuellement vide et d'un champ variable (non protégé) éventuellement vide aussi.

On peut rapprocher ces concepts aux constructeurs de types des langages de programmation typés tels que PASCAL. Ainsi aux concepts de Menu, Tableau et Sous-cadre correspondent respectivement des notions d'ensemble, tableau et structures du langage PASCAL par exemple.

## III.4. FONCTIONNALITES DU SYSTEME.

Dans la version actuelle du système on a mis l'accent sur les quatre fonctionnalités suivantes :

- Définition des types utilisateur (types prédéfinis).
- Interactivité et souplesse d'utilisation.
- Archivage et contrôle.

- Utilisation.

## III.4.1. DEFINITION DES TYPES UTILISATEUR.

La première étape de conception d'une application interactive consiste à décrire les supports de dialogue. Ces outils doivent être spécifiés dans un langage adapté à la conception de l'utilisateur.

Ainsi, pour décrire l'écran de type menu de la fig.III.1, l'utilisateur commence par décomposer son écran en trois parties: Une partie concerne le menu proprement dit en indiquant les différents choix possibles. Une deuxième partie concerne le dialogue avec l'utilisateur et sert à recueillir la réponse de l'utilisateur. Enfin, une troisième partie servira à afficher un message d'erreur en cas de réponse erronée. Ce simple exemple est très révélateur des outils attendus du système. Evidemment cette méthode de spécification trouve tout son intérêt dans le cas d'une utilisation interactive. Partant de cette idée de conception déductive et structurée on a défini dans SAGE un certain nombre de concepts facilitant le développement des applications interactives.

### III.4.1.1. L'ARTICLE.

Le concept d'article permet de spécifier l'opération de saisie d'une donnée élémentaire. Il est composé d'un champ fixe (représentant le texte accompagnateur) et un champ variable (éventuellment vide) servant à la saisie d'une donnée. En effet il est ergonomiquement souhaitable que la saisie d'une donnée s'accompagne toujours d'un libellé commentaire. Un article peut contenir simplement un champ fixe dans le cas, par exemple, de définition des options d'un menu ou le texte d'un écran d'aide. La définition d'un article réduit à un champ variable peut servir, par exemple, à l'impression d'un message d'erreur ou à la saisie d'un champ dont la signification est laissée à la découverte de l'utilisateur (programme de jeux, réponse en EAO..)

### III.4.1.2. LE SOUS-CADRE.

La définition de masque d'écrans peut souvent amener à regrouper des Articles en des entités manipulables telle quelle. Il en est ainsi du regroupement Adresse de l'écran de la fig.III.2.

Dans SAGE cette possibilité a été introduite par le concept de Sous-Cadre. Ce concept permet d'effectuer une description véritablement méthodique et déductive du dialogue. En effet, cette notion permet de concevoir un dialogue en utilisant des termes plus proches de la conception de l'utilisateur en

parlant de parties titre, menu et dialogue par opposition au concept plus abstrait de fenêtre utilisé dans d'autres systèmes [GES,80]. La description complète d'un Scadre (ou plus exactement d'une utilisation d'un Scadre cf. def. § III.4.1.5) doit d'autre part indiquer les positions respectives des Articles dans le Scadre. Cette opération est faite au niveau de l'édition.

```
APPL. GESTION BIBLIOTHEQUE

                                        option invalide
                                        xxxxxxxxxxxxxxx

                    Gestion des abonnés

        Création.....: C
        Suppression...: S
        Modification..: M
        Accés.........: A
        Fin...........: F


                              Choix : R
```

fig. III.1

```
APPL. GESTION BIBLIOTHEQUE

        Caractéristiques d'un abonné

No.abonné : .........
Nom    : ..................    Prenom : ................

Adresse :

    No. Rue : .......      Rue  : ....................
    Ville :....................
    Code postal :................

Ouvrages empruntés :

  N°     Auteur       Date emp.        Date ret.

...    ........     ../../....       ../../....
...    ........     ../../....       ../../....
...    ........     ../../....       ../../....
```

fig. III.2

### III.4.1.3. LE MENU.

Un autre concept trés utilisé dans le dialogue "pleine-page" est le concept de Menu. En effet, souvent à un stade du dialogue, il est présenté à l'utilisateur un ensemble d'options où l'utilisateur est invité à choisir une option parmi cet ensemble. Dans le dialogue ligne à ligne, l'utilisateur effectue son choix en tapant sa réponse sous forme d'une chaine. Dans le dialogue "pleine-page", l'opération de sélection utilise une autre ressource offerte par la deuxième dimension et réduit le temps et l'attention de l'utilisateur. Il existe plusieurs variantes de description d'un menu. La plus courante est celle qui consiste à présenter toutes les options, la sélection de l'utilisateur se fait soit en tapant un code correspondant à son choix soit en utilisant des touches de fonctions soit enfin en utilisant le mécanisme de désignation directe qui consiste souvent à positionner le curseur (ou tout autre moyen de sélection) sur l'option choisie et à actionner une touche de fonction.

Dans SAGE le type Menu n'a pas reçu de définition particulière puisqu'il représente un cas particulier (on dit

Sous type) du type Scadre. Un Menu est un Scadre composé d'un ensemble d'Articles de type champ fixe uniquement et d'un Article dont la partie variable sert à recueillir la réponse de l'utilisateur (éventuellement vide dans le cas de sélection par désignation directe).

### III.4.1.4. LE TABLEAU.

Le concept de tableau permet de spécifier une liste homogène de valeurs d'un type donné disposé en ligne ou en colonne. En effet il arrive souvent qu'une entrée ou sortie informatique s'exprime sous forme d'une liste de valeurs d'une variable donnée ; d'ailleurs le dialogue "pleine-page" trouve son intérêt dans cette forme de saisie dans le cas de mise à jour d'une liste de valeurs. En effet l'utilisateur peut accèder directement à la valeur qu'il veut modifier simplement en positionnant le curseur à cet endroit. En général la description d'un tableau se compose d'un en-tête sous forme de libellé et une série de valeurs en nombre fixe (cf. ex: fig.III.2).

Pour décrire un tableau d'Articles dans SAGE, l'utilisateur doit spécifier l'Article de base, dont la partie fixe est utilisée comme en-tête et la partie variable comme valeur répétitive, le nombre d'occurence d'apparition (la "dimension" du tableau), l'alignement souhaité (horizontal ou vertical) et enfin l'espacement éventuel entre les valeurs. Encore une fois un tableau d'Articles peut être décrit comme un cas particulier du Scadre. En fait cette possibilité d'instanciation, comme dans le cas des Menus, a été surtout utilisée dans l'étape de spécification (cf.chap.IV. § 2.6.).

### III.4.1.5. LE CADRE.

Le concept de Cadre (ou écran) a été initialement introduit par les applications d'EAO [IBM,69] pour décrire un état où un texte suivi d'une série de questions est présenté à l'utilisateur. Suivant la réponse de l'utilisateur, le système soit le dirige vers un autre écran soit affiche un message d'erreur et se met en attente d'une autre réponse de l'utilisateur. Actuellement tout le monde s'accorde pour utiliser le concept d'écran comme outil d'interface entre le programme d'application et l'utilisateur final. Pour le programme, il exprime un modèle d'interaction avec l'utilisateur final. Un état du dialogue correspond pour le programme soit à l'affichage de résultats, soit à la demande d'informations, soit au choix parmi un ensemble d'alternatives, soit à un écran d'aide soit enfin dans le cas le plus général à une combinaison de toutes ces possibilités. Cette structuration doit tenir compte, par ailleurs, de l'aspect ergonomique de l'utilisation [MAR,73]. Un des meilleurs moyens de concilier

ces deux aspects, qui a été adopté dans SAGE, est d'impliquer et le programmeur et l'utilisateur final dans la conception des écrans.

Un Cadre dans SAGE, consiste en un ou plusieurs composants. Un composant est soit un Article, Menu, un tableau ou enfin un Scadre.

Le regroupement des composants se fait en phase d'édition.

### III.4.1.6. NOTION D'UTILISATION.

Une de nos préoccupations majeures au niveau de la conception était de séparer les entités logiques des caractéristiques d'affichage. Cet objectif a été atteint par la notion d'utilisation (ou exemplaire). En effet la création d'un Cadre (ou un Sous-cadre) passe par deux étapes :

- La première étape consiste à donner les différentes entités logiques intervenant dans la description du Cadre.
- La deuxième étape concerne la disposition géographique des divers composants. Cette disposition permet de créer un exemplaire du Cadre ; une autre disposition créerait un deuxième exemplaire. Evidemment, pour une question de souplesse cette distinction est cachée à l'utilisateur. Un Cadre (ou un Sous-cadre) sera donc défini par un interface fixe (la liste logique des composants) et une représentation variable (la disposition géographique des composants). Le programme utilisateur ne connait du cadre que l'interface, il pourrait dans une même application passer d'une utilisation à une autre sans modification du programme. Actuellement cette différence est limitée aux représentations géographiques mais d'autres types de différences peuvent être envisagées. Ainsi par exemple, dans PIE [GOL,82],"Personnal Information Environnement" développé au centre de recherche de Xerox autour de Small-Talk où l'utilisateur peut créer pour un objet donné autant de descriptions qu'il veut combinant par exemple texte et graphique. Ces différentes descriptions peuvent être visualisées simultanément. Parmi les applications de cette technique on peut citer : le développement de logiciel où le concept de multidescriptions permet à plusieurs utilisateurs de travailler sur le même projet et les systèmes documentaires. Signalons enfin que bien que cet environnement utilise un matériel et un langage bien adapté (Small-Talk) [GOL,83],[BYT,82], il ne va pas sans créer certains problèmes méthodologiques .On pourra se reporter à ce titre à une discussion dans le même article.

### III.4.2. CONSTRUCTION ASSISTEE ET EDITION.

Une autre caractéristique importante qui permet d'utiliser de façon efficace le système SAGE est la souplesse d'utilisation. A la définition de supports de dialogue il faut ajouter le développement d'un éditeur d'écrans. En effet avant d'utiliser un écran, il faut le concevoir c.à.d décrire les composants qui le

constituent et leurs positions respectives. La façon la plus simple d'y parvenir est d'en faire un dessin sur une feuille en essayant les différentes combinaisons à l'image du typographe qui compose une page du journal en combinant plusieurs "pavés". L'éditeur d'écran permet d'effectuer les mêmes opérations interactivement en mode page :

L'utilisateur se met devant la console et "dessine" son écran avec tout le confort et la souplesse qui en résultent (la possibilité de documentation, sos...).

Pratiquement, l'utilisateur peut éditer soit un Cadre ou un Scadre.

Dans le cas d'un Cadre par exemple, l'utilisateur peut effectuer les opérations suivantes :

- Appel et copie d'un composant donné à une coordonnée donnée.
- Supprimer un composant donné.
- Déplacer un composant donné soit d'une position élémentaire dans une direction donnée soit à une coordonnée quelconque.

Insistons sur le fait que dans la phase d'édition d'un Cadre, le type de composant (Article, Sous-Cadre, Menu ou Tableau) n'a pas d'importance. L'utilisateur manipule les composants comme des pavés rectangulaires connus simplement par leur position d'origine (coin en haut à gauche) et le coin opposé (en bas à droite). En particulier, pour supprimer ou déplacer un composant, l'utilisateur positionne simplement le curseur à l'intérieur du pavé correspondant et appuie sur une touche (CTRL-S pour supprimer, CTRL-H, CTR-B, CTRL-D, CTRL-G pour déplacer d'une position élémentaire en haut,en bas, à droite ou à gauche respectivement). L'édition d'un Cadre (plus exactement un exemplaire du Cadre) se termine soit par la sauvegarde de celui-ci soit par l'abandon de l'utilisateur.

Les opérations d'édition nécessitent la construction en mémoire d'une structure intermédiaire facilitant ces opérations. L'opération de sauvegarde d'un Cadre se traduit alors par le transfert de cette structure en mémoire secondaire.

## III.4.3. ARCHIVAGE ET CONTROLES.

### III.4.3.1. ARCHIVAGE DES ENTITES.

Au sein d'une organisation informatique en vraie grandeur, le développement d'applications conversationnelles peut devenir un problème complexe du fait de la diversité des agents qui y sont impliqués. En effet dans le plus simple des cas, la simple saisie d'un bon de commande en gestion de stock par exemple, passe par au moins trois services différents. Le programmeur (le service informatique), l'utilisateur final et le service de saisie(opératrice). A ce problème d'organisation, il faut ajouter un autre problème dont souffrent beaucoup les services informatiques : le problème de réutilisabilité. En effet, les programmeurs passent souvent une grande partie de leur temps à reécrire des parties existantes ou semblables à d'autres applications.

Face à ces problèmes, il est nécessaire de faire des efforts de standardisation et de mettre en place une gestion efficace permettant aux utilisateurs de récupérer des parties de descriptions ou même des parties d'applications entières.

C'est le but poursuivi dans SAGE. En effet dans SAGE nous proposons une méthode de programmation et aussi de gestion des applications interactives dont un des objectifs est la réutilisation d'entités existantes.

En effet pour chacune des entités Article, Sous-Cadre, Tableau et Cadre l'utilisateur se construit une bibliothèque qu'il peut mettre à jour à sa guise.

L'utilisation d'une entité dans la création d'un Cadre, par exemple, passe par deux étapes : d'abord creér l'entité et l'archiver (si elle n'éxiste pas) et ensuite l'utiliser dans la définition du Cadre.

Cette façon de procéder peut paraître à priori lourde. C'est vrai dans un petit service ou une mono-utilisation. Par contre dans une grande organisation, ce moyen devient très utile notamment par la possibilté d'archiver des contrôles et l'utilisation d'une opération de duplication (qui permet de construire une nouvelle entité à partir d'une ancienne en ne gardant que les parties utiles).

Cet ensemble de bibliothèques constitue un Dictionnaire de Données (D.D). Un D.D contient, outre les entités Article, Sous-Cadre, Tableau et Cadre décrits selon le modèle entité-associations, une bibliothèque de types et de procédures (cf. § III.4.3.2). Le D.D est modifié à l'aide des opérations classiques de mise-à-jour des bases de données en respectant certaines contraintes au niveau de certaines opérations. Ces Contraintes sont liées essentiellement à la relation "est-utilisé dans" entre entités.

Pour chaque type d'entité l'utilisateur peut soit :

- ajouter une nouvelle entité, si elle n'éxiste pas, en lui associant un identifiant (son nom) qui sert de clé d'accés.

- supprimer une entité identifiée par son nom, si elle existe et si elle n'est utilisée dans aucune autre entité.

- accèder à une entité à partir de son nom complet ou à un groupe d'entité à partir d'un nom ambigü. Ainsi pour accèder à tous les articles dont le nom commence par AC, il tapera simplement AC*.

-modifier une entité donnée. Dans le cas d'une entité élémentaire, Article ou Tableau, cette opération se traduit par la modification des attributs de l'entité. Dans le cas d'une entité structurée, Sous-Cadre ou Cadre, cette opération se décompose en deux sous-opérations suivant que l'entité a une ou plusieurs utilisations (cf. définition au § III.3.8) :

i. si le Cadre (ou le Sous-Cadre) possède une seule utilisation, l'utilisateur peut modifier le groupe composant le Cadre (par adjonction ou suppression d'un ou plusieurs composants).

ii. si le Cadre possède plus d'une utilisation, les seules opérations que peut faire l'utilsateur sont : créer ou supprimer une autre utilisation en disposant autrement les différents composants. Il ne peut modifier ni le nombre ni la structure des composants.

-ou dupliquer une entité, c.à.d créer une nouvelle occurence d'une entité à partir d'une autre en modifiant les parties appropriées. Cette opération regroupe les opérations de création, d'accès et de transfert.

### III.4.3.2. SPECIFICATION DES CONTROLES.

La définition de types prédéfinis et les opérations d'édition ne concernent qu'un aspect de l'écran (on pourrait l'appeler syntaxe). Un autre aspect aussi important concerne l'interaction avec l'utilisateur. En effet la saisie d'un écran est toujours suivie d'une partie contrôle qui vérifie les réponses de l'utilisateur sur le plan de la consistance et complétude. C'est cette tâche que le module de contrôle essaie d'automatiser. Dans SAGE nous avons essayé d'intégrer au maximum la spécification de contrôles dans les descriptions de l'écran pour benéficier des avantages cités plus haut (cf. § III.3.1).

A chaque entité sera associée une partie contrôle. Dans le cas d'une entité élémentaire (Article), la spécification des contrôles se compose de deux parties : une première partie qu'on peut qualifier de statique, décrit l'image de la partie

variable à l'aide d'un format analogue au PICTURE de COBOL. Une deuxième partie portant sur la valeur de la zone, est décrite en combinant constructeurs de types et prédicats (cf. §II.2.2 et § IV.2.8.2).

Ainsi pour décrire les contrôles du type Date d'emprunt, l'utilisateur spécifie d'abord le format sous la forme:NNXNNXNNN

Ensuite pour la partie type il donnera simplement le type Date en supposant que Date est un type déja décrit.

Le type Date comme d'autres types plus élémentaires : Entier, Réel, Chaine, ainsi que les types intervalle, structures et ensemble sont des constructeurs qui auront été déja défini dans le Dictionnaire de Données.

Pour une entité structurée le type associé est obtenu par composition des types de ses composants. Parallèlement à cette composition l'utilisateur peut spécifier des contrôles sous forme de prédicats :

ex :

```
Adresse : struct (nrue : Entier  ; rue,ville : chaine ;
                  codpos:Entier)
avec
   nrue < 1000
  codpos <> 0
```

Le type associé à une entité structurée dépend de sa nature. Aux entités Scadre, Tableau et Menu sont associées respectivement les types structure, tableau et ensemble. Mais l'utilisateur peut aussi utiliser des types prédéfinis déja existant dans le Dictionnaire de Données.

En fait ce mécanisme de composition est directement lié à celui de l'interprétation présenté au paragraphe suivant.

### III.4.4. UTILISATION.

Cette phase concerne l'exploitation des écrans dans un programme utilisateur. Tous les concepts définis précédements ont pour but de simplifier cette tâche. En effet le but poursuivi dans SAGE est de faciliter la programmation des applications interactives en laissant tous les détails du dialogue au système. Concrètement ceci se traduit au niveau de l'application par :

a) limiter les procédures d'échange entre le programme utilisateur et le système. Plus précisement le programmeur dispose de seulement deux procédures principales d'échange :

i.la procédure "affiche-écran" qui affiche un écran, dont le

nom et l'utilisation sont donnés en paramètres à cette procédure. Cette procédure a aussi pour effet d'initialiser toutes les parties variables de l'écran à vide si c'est le premier affichage de cet écran, aux dernières valeurs saisies si non.

ii.la procédure "lec-écran" qui suit immediatement l'appel de la procédure d'affichage. Cette procédure saisit les parties variables du Cadre en affichage, exécute les contrôles spécifiés au moment de la description du Cadre et produit un tampon formaté directement exploitable par le programme utilisateur.

b) garder la logique de l'application. Les opérations d'édition, l'affichage en particulier, nécessitent la construction en mémoire d'une structure adaptée facilitant ces opérations mais rend difficile le passage aux structures manipulées par le programme utilisateur qui s'expriment à l'aide des types classiques de langages de programmation : variable simple, structure, tableau..etc. Pour prendre en compte ce problème,lors de la description d'un Cadre le système génère l'équivalent de la description du Cadre dans le langage de programmation de l'utilisateur. Concrètement le système génère la description d'un type RECORD en PASCAL. Parallèlement à ce type, il est généré une procédure d'interprètation de ce type qui réalise le passage de la chaine de caractères saisie à la description du type proprement dite.

Cette procédure ainsi que le type générée sont archivés dans le Dictionnaire de Données.

Ainsi pour le Cadre de la fig.III.2 on obtient pour le type et la procédure générée.

```
(* -----   description du cadre ES  ---- *)

ES = record
       TNUM : array [1.. 5] of integer ;
       SCO1 : record
                NOM : string (20) ;
                PRENOM : string (20) ;
                ADRESSE : string (30) ;
              end ;
     end;


(* -----   procedure d'interpretation du cadre ES  ---- *)

procedure SA_LECT(var SA_VAL : string[255] ; var VALREC : ES);
var
  SA_I,SA_J,SA_LONG : integer;
  SA_TRA            : string [80];
procedure AFFEC(var STR : string);
begin
  STR:='';
  while SA_VAL[SA_I] <> chr($00) do
  begin
    STR:=concat(STR,SA_VAL[SA_I]);
    SA_I:=SA_I+1
  end;
end;
begin
  SA_LONG:=length(SA_VAL);
  SA_I:=1;
  with VALREC do
  begin
    for SA_J:=1 to 5 do
    begin
      AFFEC(SA_TRA);
      CONVNUM(SA_TRA,TNUM[SA_J]);
      SA_I:=SA_I+1
    end;
    with SCO1 do
    begin
      AFFEC(NOM) ;
      SA_I:=SA_I+1;
      AFFEC(PRENOM) ;
      SA_I:=SA_I+1;
      AFFEC(ADRESSE);
      SA_I:=SA_I+1;
    end
  end
end;
```

## III.5. EXEMPLE D'UTILISATION.

Dans cette section nous allons donner la marche à suivre pour décrire un Cadre et l'utiliser dans une application. Mais auparavant nous donnons quelques précisions générales sur le système.

### III.5.1. LE POSTE DE TRAVAIL.

Il se compose, dans la version actuelle du système, de deux consoles (voir format fig.III.3) :

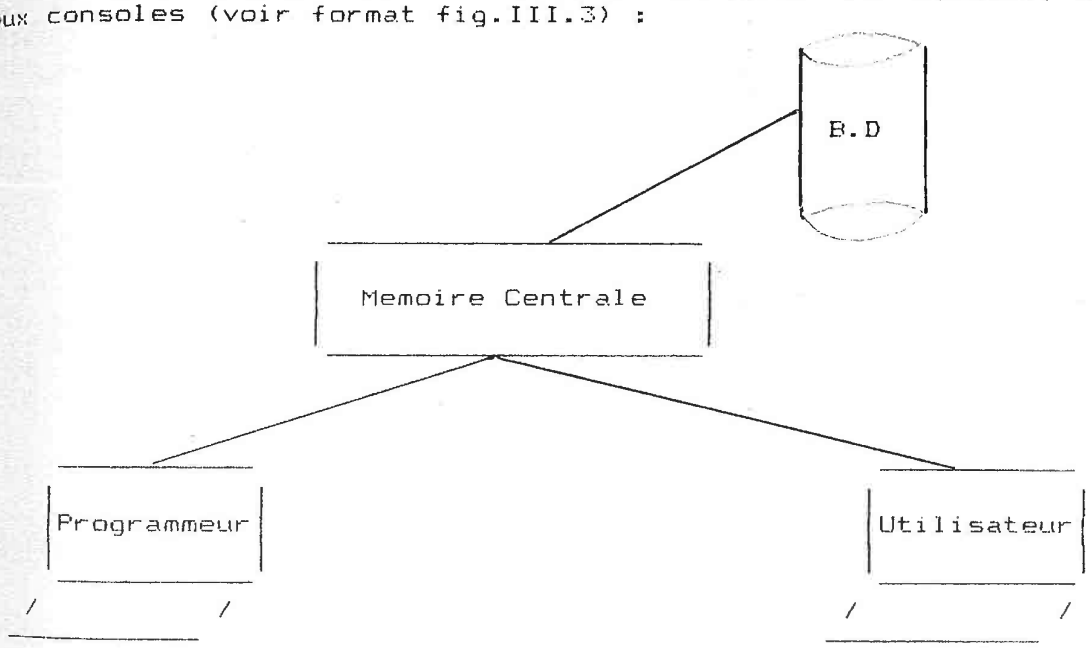



fig. III.3

-l'écran "programmeur" ou de pilotage sert d'écran de dialogue et à afficher des information d'aide (liste des éléments d'une bibliothèque, liste des commandes disponibles, rôle de chaque touche de fonction, et des messages divers (erreurs, commentaires), ...).
Actuellement cet écran est l'écran standard de la REE qui est relié au micro-ordinateur.

-L'écran "utilisateur" sert de support à la construction des différentes entités. Le terminal actuellement utilisé et une TAB132/15 dont les caractèristiques sont :

Ecran :

Norme ANSI.
Taille (80 ou 132) X 24
possibilité de caractères semi-graphiques.

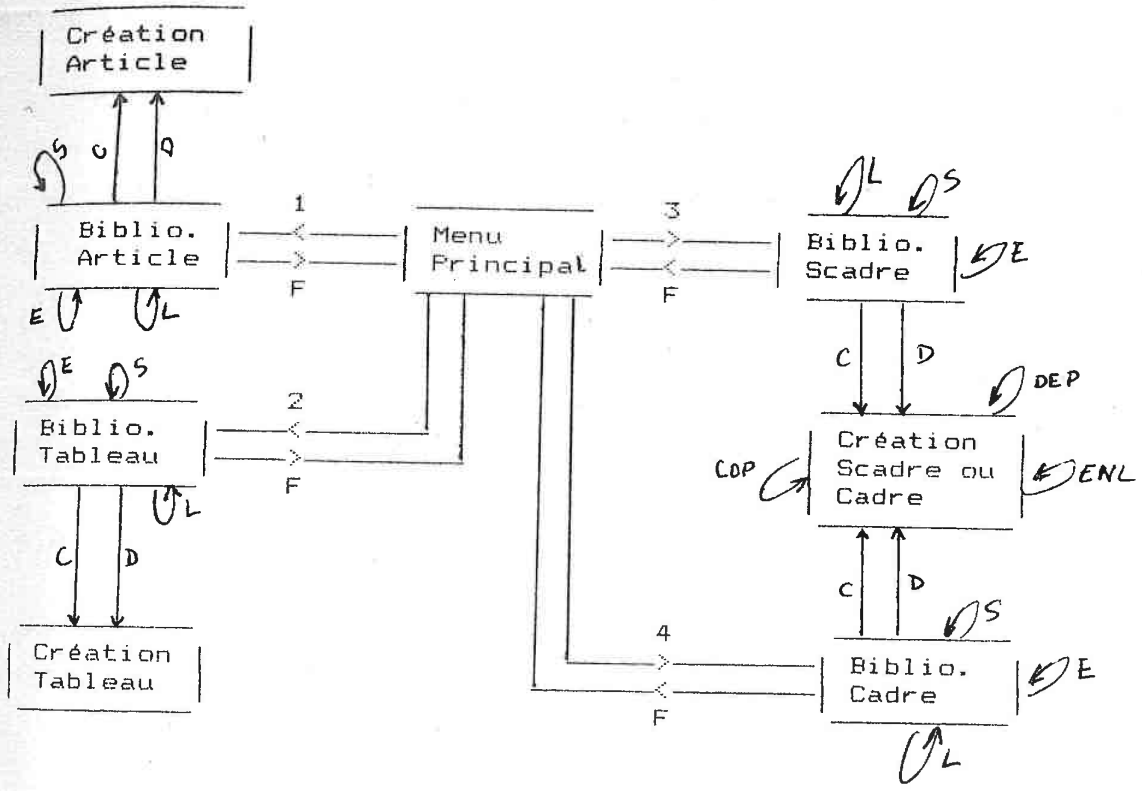
Clavier :

possibilté de touches programmables.

Rappelons que pour un objectif de portabilité, seules les caractéristiques de base du terminal (adressage du curseur, attributs visuels,....) sont utilisées, les autres fonctions (mode insertion, effacement de caractères ou de lignes, mode page ...) ont été programmées. Ces caractéristiques sont par ailleurs stockées dans une table ce qui rend très facile leur modification dans le cas d'utilisation d'un autre terminal.

## III.5.2. DESCRIPTION D'UNE SESSION.

Pour utiliser le système l'utilsateur a le choix entre dialogue par commandes ou sélection d'une liste. Dans ce dernier cas les commandes disponibles sont affichées ous forme de menus successifs dont le graphe de la fig.III.4 en donne l'enchaînement :




Légende :

C : Création
S : Suppression
L : Lecture
E : etat bibliotèque
D : Duplication

COP : copie d'un élément
DEP : Déplacer un élément
ENL : enlever un élément
F : Fin

fig. III.4

Pour sélectionner une commande, l'utilisateur positionne simplement le curseur à l'endroit de cette commande et appuie sur la touche CTRL-F. Dans le cas de dialogue par commandes, la syntaxe d'une commande est de la forme :

<ordre><param1><param2><param3>

Les ordres possibles sont :

```
          ART
- CRE    /_SCA                    "création d'une entité".
         \ CAD

          ART
- MAJ    /_SCA                    "archivage d'une entité"
         \ CAD

          ART
- LIR    /_SCA                    "accès à une entité"
         \ CAD

          ART
- DUP    /_SCA                    "dupliquer une entité"
         \ CAD

          ART
- ETA    /_SCA                    "lister la bibliothèque entité"
         \ CAD

              ART
- COP,DEP    /                   "copier ou déplacer une entité
             \  SCA              "à une coordonnée donnée"

- SOS,EFF,GOTO,...                "fonctions d'aides..."
```

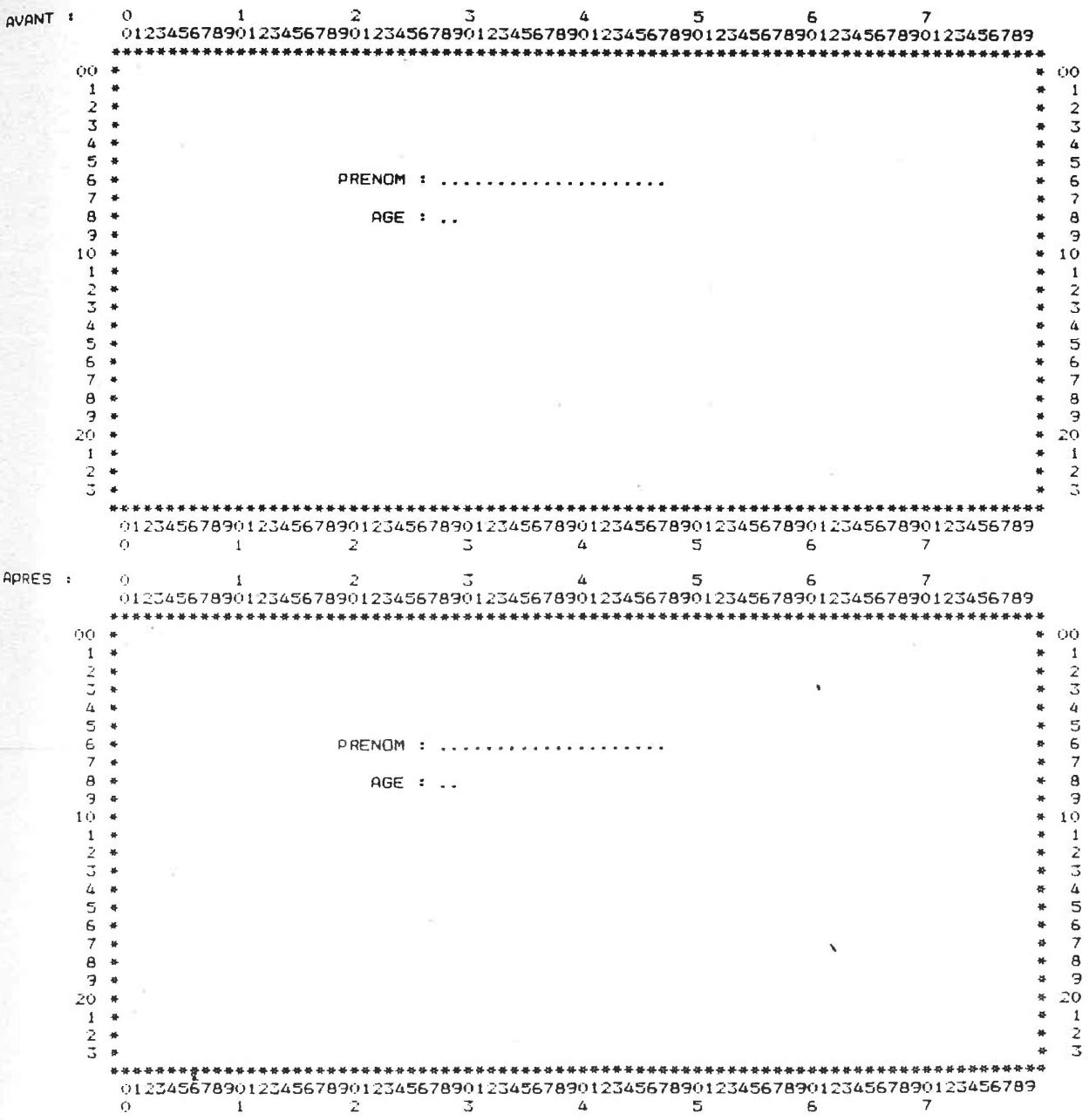
Pour une description précise du rôle de chaque commande, on se reportera au manuel d'utilisation dans [BOC,84].

On présente ci-dessous les différents phases de création d'un Cadre et l'effet de chaque commande sur les deux types d'écrans:

```
GRAN PROGRAMMEUR
DEBUT :    0         1         2         3         4         5         6         7
           01234567890123456789012345678901234567890123456789012345678901234567890123456789
           ********************************************************************************
                                                                                        * 00
        00 *LIR ART A1                                                                  *  1
         1 *ooération en cours                                                          *  2
         2 *--------------------------------------------------------------------------- *  3
         3 *                                                                            *  4
         4 *                                                                            *  5
         5 *                                                                            *  6
         6 *                                                                            *  7
         7 *                                                                            *  8
         8 *                                                                            *  9
         9 *                                                                            * 10
        10 *                                                                            *  1
         1 *                                                                            *  2
         2 *                                                                            *  3
         3 *                                                                            *  4
         4 *                                                                            *  5
         5 *                                                                            *  6
         6 *                                                                            *  7
         7 *                                                                            *  8
         8 *                                                                            *  9
         9 *                                                                            * 20
        20 *                                                                            *  1
         1 *                                                                            *  2
         2 *                                                                            *  3
         3 *--------------------------------------------------------------------------- *
           ********************************************************************************
            01234567890123456789012345678901234567890123456789012345678901234567890123456789
            0         1         2         3         4         5         6         7

FIN :       0         1         2         3         4         5         6         7
            01234567890123456789012345678901234567890123456789012345678901234567890123456789
           ********************************************************************************
                                                                                        * 00
        00 *LIR ART A1                                                                  *  1
         1 *ooération terminée                                                          *  2
         2 *--------------------------------------------------------------------------- *  3
         3 *                                                                            *  4
         4 *                                                                            *  5
         5 *libellé de la partie fixe                                                   *  6
         6 *NOM :                                                                       *  7
         7 *                                                                            *  8
         8 *                                                                            *  9
         9 *                                                                            * 10
        10 *format de la partie variable :                                              *  1
         1 *A20                                                                         *  2
         2 *                                                                            *  3
         3 *                                                                            *  4
         4 *                                                                            *  5
         5 *zone de documentation libre :                                               *  6
         6 *article permettant la saisie d' un nom dans le programme ESSAI              *  7
         7 *                                                                            *  8
         8 *                                                                            *  9
         9 *                                                                            * 20
        20 *                                                                            *  1
         1 *                                                                            *  2
         2 *                                                                            *  3
         3 *--------------------------------------------------------------------------- *
           ********************************************************************************
            01234567890123456789012345678901234567890123456789012345678901234567890123456789
            0         1         2         3         4         5         6         7
```

ECRAN UTILISATEUR



ECRAN PROGRAMMEUR

```
DEBUT :    0         1         2         3         4         5         6         7
           01234567890123456789012345678901234567890123456789012345678901234567890123456789
           ********************************************************************************
        00 *COP 4 19 4 28                                                               * 00
         1 *opération en cours                                                          *  1
         2 *-----------------------------------------------------------------------------*  2
         3 *                                                                            *  3
         4 *                                                                            *  4
         5 *libellé de la partie fixe                                                   *  5
         6 *NOM :                                                                       *  6
         7 *                                                                            *  7
         8 *                                                                            *  8
         9 *                                                                            *  9
        10 *format de la partie variable :                                              * 10
         1 *A20                                                                         *  1
         2 *                                                                            *  2
         3 *                                                                            *  3
         4 *                                                                            *  4
         5 *zone de documentation libre :                                               *  5
         6 *article permettant la saisie d' un nom dans le programme ESSAI              *  6
         7 *                                                                            *  7
         8 *                                                                            *  8
         9 *                                                                            *  9
        20 *                                                                            * 20
         1 *                                                                            *  1
         2 *                                                                            *  2
         3 *-----------------------------------------------------------------------------*  3
           ********************************************************************************
           01234567890123456789012345678901234567890123456789012345678901234567890123456789
           0         1         2         3         4         5         6         7

FIN :      0         1         2         3         4         5         6         7
           01234567890123456789012345678901234567890123456789012345678901234567890123456789
           ********************************************************************************
        00 *COP 4 19 4 28                                                               * 00
         1 *opération terminée                                                          *  1
         2 *-----------------------------------------------------------------------------*  2
         3 *                                                                            *  3
         4 *                                                                            *  4
         5 *libellé de la partie fixe                                                   *  5
         6 *NOM :                                                                       *  6
         7 *                                                                            *  7
         8 *                                                                            *  8
         9 *                                                                            *  9
        10 *format de la partie variable :                                              * 10
         1 *A20                                                                         *  1
         2 *                                                                            *  2
         3 *                                                                            *  3
         4 *                                                                            *  4
         5 *zone de documentation libre :                                               *  5
         6 *article permettant la saisie d' un nom dans le programme ESSAI              *  6
         7 *                                                                            *  7
         8 *                                                                            *  8
         9 *                                                                            *  9
        20 *                                                                            * 20
         1 *                                                                            *  1
         2 *                                                                            *  2
         3 *-----------------------------------------------------------------------------*  3
           ********************************************************************************
           01234567890123456789012345678901234567890123456789012345678901234567890123456789
           0         1         2         3         4         5         6         7
```

ÉCRAN UTILISATEUR

AVANT :

```
    0         1         2         3         4         5         6         7
    01234567890123456789012345678901234567890123456789012345678901234567890123456789
   ********************************************************************************
00 *                                                                              * 00
 1 *                                                                              *  1
 2 *                                                                              *  2
 3 *                                                                              *  3
 4 *                                                                              *  4
 5 *                                                                              *  5
 6 *                      PRENOM : .....................                          *  6
 7 *                                                                              *  7
 8 *                         AGE : ..                                             *  8
 9 *                                                                              *  9
10 *                                                                              * 10
 1 *                                                                              *  1
 2 *                                                                              *  2
 3 *                                                                              *  3
 4 *                                                                              *  4
 5 *                                                                              *  5
 6 *                                                                              *  6
 7 *                                                                              *  7
 8 *                                                                              *  8
 9 *                                                                              *  9
20 *                                                                              * 20
 1 *                                                                              *  1
 2 *                                                                              *  2
 3 *                                                                              *  3
   ********************************************************************************
    01234567890123456789012345678901234567890123456789012345678901234567890123456789
    0         1         2         3         4         5         6         7
```

APRES :

```
    0         1         2         3         4         5         6         7
    01234567890123456789012345678901234567890123456789012345678901234567890123456789
   ********************************************************************************
00 *                                                                              * 00
 1 *                                                                              *  1
 2 *                                                                              *  2
 3 *                                                                              *  3
 4 *                         NOM : .....................                          *  4
 5 *                                                                              *  5
 6 *                      PRENOM : .....................                          *  6
 7 *                                                                              *  7
 8 *                         AGE : ..                                             *  8
 9 *                                                                              *  9
10 *                                                                              * 10
 1 *                                                                              *  1
 2 *                                                                              *  2
 3 *                                                                              *  3
 4 *                                                                              *  4
 5 *                                                                              *  5
 6 *                                                                              *  6
 7 *                                                                              *  7
 8 *                                                                              *  8
 9 *                                                                              *  9
20 *                                                                              * 20
 1 *                                                                              *  1
 2 *                                                                              *  2
 3 *                                                                              *  3
   ********************************************************************************
    01234567890123456789012345678901234567890123456789012345678901234567890123456789
    0         1         2         3         4         5         6         7
```

ECRAN PROGRAMMEUR

DEBUT :

```
          0         1         2         3         4         5         6         7
          01234567890123456789012345678901234567890123456789012345678901234567890123456789
         ********************************************************************************
      00 *MAJ SCA SC01 2                                                              * 00
       1 *opération en cours                                                          *  1
       2 *-----------------------------------------------------------------------------*  2
       3 *                                                                            *  3
       4 *                                                                            *  4
       5 *libellé de la partie fixe                                                   *  5
       6 *NOM :                                                                       *  6
       7 *                                                                            *  7
       8 *                                                                            *  8
       9 *                                                                            *  9
      10 *format de la partie variable :                                              * 10
       1 *A20                                                                         *  1
       2 *                                                                            *  2
       3 *                                                                            *  3
       4 *                                                                            *  4
       5 *zone de documentation libre :                                               *  5
       6 *article permettant la saisie d' un nom dans le programme ESSAI              *  6
       7 *                                                                            *  7
       8 *                                                                            *  8
       9 *                                                                            *  9
      20 *                                                                            * 20
       1 *                                                                            *  1
       2 *                                                                            *  2
       3 *-----------------------------------------------------------------------------*  3
         ********************************************************************************
          01234567890123456789012345678901234567890123456789012345678901234567890123456789
          0         1         2         3         4         5         6         7
```

FIN :

```
          0         1         2         3         4         5         6         7
          01234567890123456789012345678901234567890123456789012345678901234567890123456789
         ********************************************************************************
      00 *MAJ SCA SC01 2                                                              * 00
       1 *élément comolétement écrit                                                  *  1
       2 *-----------------------------------------------------------------------------*  2
       3 *                                                                            *  3
       4 *                                                                            *  4
       5 *libellé de la partie fixe                                                   *  5
       6 *NOM :                                                                       *  6
       7 *                                                                            *  7
       8 *                                                                            *  8
       9 *                                                                            *  9
      10 *format de la partie variable :                                              * 10
       1 *A20                                                                         *  1
       2 *                                                                            *  2
       3 *                                                                            *  3
       4 *                                                                            *  4
       5 *zone de documentation libre :                                               *  5
       6 *article permettant la saisie d' un nom dans le programme ESSAI              *  6
       7 *                                                                            *  7
       8 *                                                                            *  8
       9 *                                                                            *  9
      20 *                                                                            * 20
       1 *                                                                            *  1
       2 *                                                                            *  2
       3 *-----------------------------------------------------------------------------*  3
         ********************************************************************************
          01234567890123456789012345678901234567890123456789012345678901234567890123456789
          0         1         2         3         4         5         6         7
```

## III.6 ORGANISATION D'UN PROGRAMME SOUS SAGE.

Dans cette section on va donner le "squelette" général d'un programme d'application interactive pleine-page utilisant les outils décrits précédement. La fig III.5 décrit le schéma fonctionnel du système en distinguant les deux phases :
définition des entités et utilisation dans un programme PASCAL.

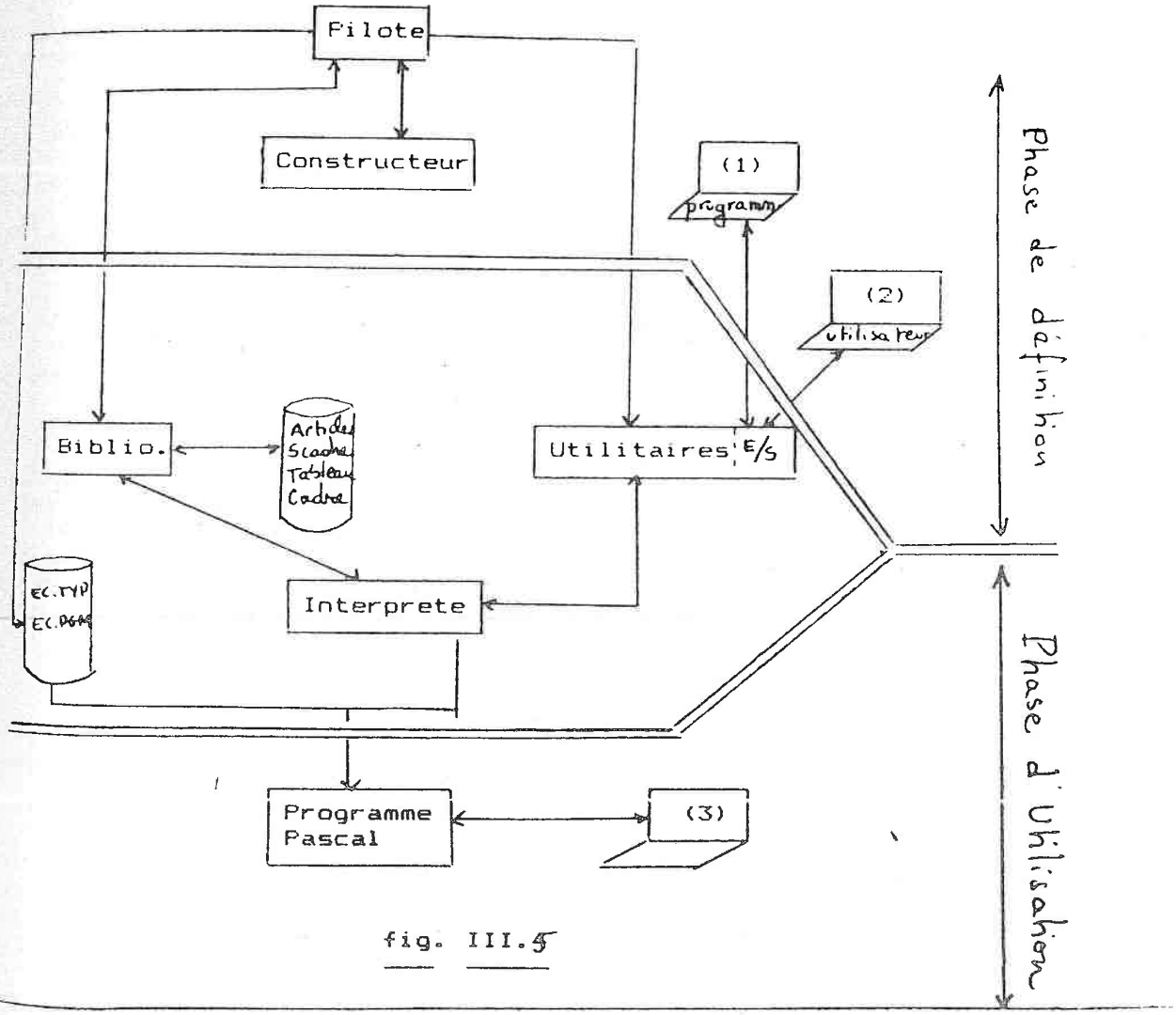



fig. III.5

Le programme demo_sage donne une exemple d'utilisation avec un seul cadre :

```
program demo-sage;

type

  (* bibliothèque des types incluant ecr_demo *)
(*$I(nclude) BIBECR.TYP *)

var
  ec      : ecr_demo;    (* une variable de type ecr_demo *)
  ut      : integer;     (* numéro d'utilisation voulue   *)
  codret  : integer;     (* code retour: 0 bien passé     *)
  err       : boolean;

external procedure aff_ecr (noecr:string;util:integer;
                            var cr:integer            );
external procedure lec_ecr (var res:boolean);

 (* biblio. des proc. d'interprètation incluant int_ecr_demo *)
(*$I BIBECR.PGM *)

begin

   (* afficher l'écran "ecr-demo" avec l'utilisation "ut" *)

  aff_ecr('ecr_demo',ut,codret);
 if codret <> 0 then err:=true (* ecran ou utilisation non
 else                           trouvé *)
 begin
   lec_ecr(err); (* lecture des zones *)
   int_ecr_demo(ec); (* récupération de l'enregistrement *)
 end
end.
```

En passant à un programme avec plusieurs cadres et en utilisant la commande par table de transitions (cf. §II.2.5) on obtient le schéma de programme suivant :

```
Program demo_dialogue;

type
(* I BIBECR.TYP *)

(* + types utilisateurs *)
etat = record
         ne       : integer;    (* numéro de l'état *)
         ecr_asoc : string;     (* écran associé    *)
         ut       : integer;    (* et utilisation   *)
       end;

var
  etat_initial,etat_final  : etat;
(* I BIBECR.EXT   *)    (* procedures externes *)
procedure Enchainement;

var
 etat_courant : etat;
 rep:choix;
begin
 etat_courant:=etat_initial;
 while etat_courant <> etat_final do ;
 begin
   rep:=Traiter(etat_courant);
   etat_courant:=Transition(etat_courant,rep);
 end;
end;

function Traiter(e:etat):choix;

var
 err    : boolean;
 codret : integer;
 c      : choix;
```

```
begin
  while not(err) do
  begin
    aff_ecr(et.ecr_asoc,et.util,codret);
    c:=lec_ecr(err);
    interp_ecr(ecr);
  end;
  Traiter:=c;
  (* enregistrer *)
end;
begin
  (* programme principal : initialisation de Transition *)
end.
```

Remarquons que la procédure Enchainement peut être utilisée telle quelle dans le programme utilisateur, ainsi qu'une grande partie de la procédure Traiter (en fait seule la procédure "enregistrer" est à définir pour chaque programme en fonction des traitements dans chaque état). L'utilisateur "instanciera" son programme simplement en détaillant pour chaque écran la procédure "enregistrer" et en initialisant la table de transitions. Parmi les développements ultérieurs du système, nous comptons réaliser un "générateur de dialogue" permettant d'automatiser la première tâche. Un autre objectif du "générateur de dialogue" est de prouver, à priori, la consistance du dialogue à l'aide de maquettes.

# CHAP.IV. SPECIFICATION DU SYSTEME SAGE.

## VI.1. GENERALITES SUR LES SPECIFICATIONS FORMELLES.

### IV.1.1. INTRODUCTION - JUSTIFICATION.

Les techniques de spécifications formelles sont apparues à la suite de certaines réflexions sur le développement de logiciels. Ces réflexions portaient sur les deux aspects suivants :

i. Il est de plus en plus admis que la résolution d'un problème informatique dépend beaucoup de la manière dont est enoncé ce problème. Ce problème devient critique dans certaines applications informatiques où le dialogue entre utilisateur et informaticien se fait à l'aide de cahier de charges dont on connait bien les inconvénients (langage naturel, sur et sous spécification, bruit et silence....) [FIN,79].[MEY,79].

ii. Les logiciels deviennent de plus en plus complexes à maintenir. Les programmeurs concentraient souvent leur attention sur la phase de codage alors que la phase clé se trouve dans la conception. Rappelons que pour ces logiciels, e nviron 60 % des ressources humaines et matérielles sont consacrées à leur maintenance [WIN,79].

Devant ces difficultés de concevoir, comprendre et maintenir des programmes; on a cherché à developper des outils de spécifi-

tation statique.

La première conception d'un programme consiste à spécifier l'univers des objets du programme en termes de types abstraits. La spécification doit se limiter à décrire simplement le comportement des types indépendamment de tous détails de réalisation. Cette abstraction signifie qu'il doit être possible de remplacer dans un programme un module par un autre, supposé avoir le même comportement, sans aucune autre modification du programme.

Un autre intérêt des spécifications est que le programme ainsi spécifié répond à un modèle mathématique rigoureux ou l'on peut faire des preuves, poser des questions (et eventuellement y répondre) avant sa réalisation.

## IV.1.2. CARACTERISTIQUES D'UNE SPECIFICATION FORMELLES.

### IV.1.2.1. ABSTRACTION.

La spécification d'un type abstrait doit se limiter à décrire simplement le comportement du type indépendamment de tout détail de réalisation. Précisement une spécification doit laisser un large éventail de choix sur la représentation du type.

Pour illustrer cette caractéristique, prenons l'exemple du problème d'affectation des étudiants dans des salles, tiré de JONES [JON,76]. Pour résoudre ce problème, l'utilisation d'un tableau indicé par les numéros de salles conviendrait. Mais il est dangereux de fonder une conception directement sur une représentation. En effet, cette représentation peut ne pas convenir (ou se revéler non performante) pour implémenter des opérations telle que : "connaitre le nombre d'étudiants dans une salle". Dans une première étape, il est plus prudent de décrire simplement les opérations portant sur le type (ici le système Palcement) qui à l'aide de spécification pré/ post conditions et en utilisant le formalisme ensembliste ("mapping sets") [JON,76], [ABR,80] s'exprimeraient par :

```
type Placement : Nom_etud    ——> Num_salle

   init     :                                  ——> Placement
      post(init(p))= (p=[])
      ———
      ; Placement vide

   placer   : Placement X Noe X Nos  ——> Placement
      pre placer(p,ne,ns) = (ne ∈/ dom(p))
      ———
      post placer(p,ne,ns,p') = (p' = p + [ne -> ns])
      ———
      ; placer un étudiant non inscrit
      ; + est la somme de deux fonctions
      ; [ne ——> ns] est la fonction à un élément

   changer  : Placement X Noe X Nos   ——> Placement
      pre changer(p,ne,ns) = (ne ∈ dom(p))
      ———
      post changer(p,ne,ns,p') = (p'=p + [ne -> ns])
      ———
      ; changer la salle d'un étudiant déja inscrit

   trouver  : Placement X Noe         ——> Nos
      pre trouver(p,ne) = (ne ∈ dom(p))
      ———
      post trouver(p,ne,res) = (res = p(ne))
      ———
      ; trouver le num. de salle de l'étudiant ne
      ; résultat detérminé par application de la fonction p à ne
ftype
———
```

IV.1.2.2. CARACTERE STATIQUE.

Tout système peut être décrit comme un ensemble d'états avec des procédures de changement d'états. Néanmoins, un état peut être défini sans faire référence à ces procédures, c.à.d qu'une information sur un état peut être utilisée sans s'occuper du fait comment elle a été créee. Ceci revient à dire que la spécification ne manipule que des constantes. Cette staticité est une autre caractéristique importante des spécifications formelles. Ainsi dans l'écriture suivante :

```
p'=placer(p,ne,ns)
```

p et p' sont considérés comme deux objets différents du type.

## IV.1.3. DEUX METHODES DE SPECIFICATION DE TYPES ABSTRAITS.

Nous allons passer en revue deux approches de spécification de types abstraits : l'approche axiomatique et algèbrique en insistant plus sur la deuxième approche puisque c'est cette dernière qui a été adoptée dans la spécification du système SAGE.

### IV.1.3.1. L'APPROCHE AXIOMATIQUE.

On la qualifie aussi d'opérationnelle ou à pré/post conditions. Elle sert souvent à expliciter un premier énoncé à partir d'une idée intuitive du problème ou du type à spécifier. On en a vu un premier exemple lors de la spécification du problème d'affectation. La spécification utilise des constructeurs primitifs. On va expliciter cette technique sur un autre exemple : le type Pile d'entiers :

```
Type Pile
invariant
  est_suite(Pile)
opérations
  init     :   ()          ----> Pile
           ; post(init) = (p =<>)

  empiler  : Pile X Ent    ----> Pile
           ; post(empiler(p,e,p')) = (p'=p*e)
  depiler  : Pile          ----> Pile
           ; pre(depiler(p)) = non (vide(p))
           ; post(depiler(p,p')) = (p'=debut(p))
  sommet   : Pile          ----> Ent
           ; pre(sommet(p)) = non (vide(p))
           ; post(sommet(p,s)) = (s=der(p))
  vide     : Pile          ----> Bool
           ; post(vide(p)) = (p = <>)
ftype
```

La définition du type Pile fait apparaitre des opérations plus générales du type suite dont le type Pile est un sous-type.

### IV.1.3.2. L'APPROCHE ALGEBRIQUE.

Cette méthode, appelée aussi implicite, a été développée par GUTTAG [GUT,78] et le groupe ADJ [GOG,77]. Pour montrer ces caractéristiques par rapport à l'approche axiomatique, nous allons reprendre l'exemple du type Pile d'entiers :

```
Type Pile

opérations

  (c)   init    : ()              ----> Pile
  (c)   empiler : Pile X Ent      ----> Pile
  (s)   depiler : Pile            ----> Pile
  (o)   sommet  : Pile            ----> Ent
  (o)   vide    : Pile            ----> Bool

axiomes

  p : Pile ; x : Ent

  1)  vide(init)=vrai

  2)  vide(empiler(p,x))=faux

  3)  depiler(empiler(p,x))=p
  4)  sommet(empiler(p,x))=x

préconditions

  pré depiler(p) = pre (sommet(p)) = non (vide(p))

ftype
```

La spécification d'un type abstrait dans ce formalisme consiste en deux parties : - la partie opérations où l'on donne le profil (domaines et co-domaines) des opérations utilisées. Ces opérations se subdivisent en trois ensembles : les constructeurs (init et empiler), simplificateurs (depiler) et observateurs(sommet,vide). - La partie axiomes où l'on décrit les relations entre ces opérations. Contrairement aux spécifications axiomatiques, on ne décrit pas ce que c'est un objet du type mais simplement comment passer d'un objet à un autre. C'est en ce sens que cette aproche peut sembler plus abstraite (implicite) que l'approche axiomatique.

### IV.1.3.3 FORME GENERALE D'UNE SPECIFICATION ALGEBRIQUE.

La forme générale d'une spécification algèbrique d'un type abstrait se présente ainsi :

type Typdd(param)

param possède liste_opér

opérations

liste des opérations(constructeurs,simplificateurs et observateurs) avec leur profil.

axiomes

Liste des équations récursives caractèrisant les opérations de simplification et d'observation sur les constructeurs.

préconditions

liste des préconditions sur certaines opérations

ftype

## IV.1.4. MECANISMES DE CONSTRUCTION DE TYPES ABSTRAITS.

Souvent les types abstraits sont définis à partir d'autre types selon des mécanismes qu'on présente ci-dessous :

### IV.1.4.1. RESTRICTION / ENRICHISSEMENT.

Ce mécanisme permet de définir un type comme un sous-type d'un autre en restreignant ou enrichissant certaines opérations. C'est le cas par exemple du type File de l'exemple du § IV.1.3.1.

Ce mécanisme est très utile dans la spécification d'un système complexe puisqu'il permet de définir et d'utiliser des types hiérarchiques et d'avoir une construction structurée et descendante. Ce mécanisme a été systèmatiquement utilisé dans la spécification du système SAGE.

### IV.1.4.2. PARAMETRISATION DES TYPES ABSTRAITS.

Il est souvent utile de définir un type paramètré par un autre type. Le type File d'entiers peut être considéré comme une

instance du type plus général Pile de Val dont la spécification reste presque identique :

```
Type Pile(Val)

opérations
   init    : ()            ------> Pile
   empiler : Pile X VAl    ------> Pile
   depiler : Pile          ------> Pile
   sommet  : Pile          ------> Val
   vide    : Pile          ------> Bool
axiomes
   p : Pile ; v : Val

1) vide(init)=vrai

2) vide(empiler(p,v))=faux

3) depiler(empiler(p,v))=p
4) sommet(empiler(p,v))=v

préconditions
pré (depiler(p)) = pré (sommet(p)) = non (vide(p))

ftype
```

Plus que des types abstraits ce mécanisme permet de définir des schémas de types abstraits et ajoute au spécification un autre niveau d'abstraction. La paramètrisation de types abstraits pose le problème de contrôle de types. Dans le type Pile il n'y a aucune restriction sur le paramètre mais souvent le paramètre doit possèder un certain nombre d'opérations utilisées dans la spécification du type.

A ces deux mécanismes il faut ajouter les opérations ensemblistes (produit cartésien, projection, union.....).

### IV.1.5. SEMANTIQUE D'UNE SPECIFICATION.

Sur une spécification formelle on peut se poser au moins deux questions. La première est de savoir si elle est consistante, c.à.d qu'elle ne contient pas de contradictions. La deuxième est de savoir si elle est complète, c.à.d est ce que les opérations et les axiomes suffisent pour décrire complètement le comportement de l'objet vis-à-vis des questions qui s'y

rapportent.

### IV.1.5.1. CONSISTANCE D'UNE SPECIFICATION.

Une spécification est consistante si elle ne contient pas de contradictions. Ainsi si nous reprenons la spécification du type Pile en ajoutant l'axiome suivant :

6) vide(depiler(p))=faux

on obtient une spécification inconsistante. En effet :

vide(depiler(empiler(init,x)) =
3) ===> = vide(init)=
1) ===> = vrai

or 6) donne faux d'où contradiction.

Comme en logique la question de consistance est résolue par la recherche d'un modèle. Un système formel est consistant s'il admet un modèle.

### IV.1.5.2. COMPLETUDE D'UNE SPECFIFICATION.

Savoir qu'une spécification est consistante n'est en général pas suffisant. Ainsi bien que la spécification précédente du type Pile soit consistante on ne sait comparer deux termes tels que:

empiler(empiler(p,x1),x2) et empiler(empiler(p,x2),x1)

En fait on ne peut démontrer ni l'égalité ni l'inégalité des deux termes ce qui est la définition de l'incomplétude.

En terme de modèle, une spécification incomplète signifie qu'elle admet des modèles non isomorphes. Dans ce cas plusieurs approches sont envisageables :

- On peut décider que deux termes sont différents s'il on ne peut démontrer leur égalité. Ceci revient à définir la sémantique d'un type algèbrique comme étant l'algèbre initiale (qui contient donc le moins d'éléments).
- Considérer comme égaux deux termes qui ne peuvent être démontré différents et on obtient dans ce cas une algèbre terminale.

### IV.1.5.3.COMPLETUDE SUFFISANTE D'UNE SPECIFICATION.

Savoir qu'une spécification algèbrique est complète n'est ni toujours réalisable ni même souhitable. Souvent la question de complètude est résolue par une notion moins forte : la complètude

suffisante introduitee par GUTTAG [GUT,78].

Soit C le sous ensemble d'un type algèbrique T dont le codomaine est la sorte d'intérêt ; dans l'exemple du type Pile C=[init,empiler] et O son complémentaire. T est suffisament complet si pour tout terme de la forme f(x1,...,xn) où f ∈ O,il existe un théorème démontrable, à partir des axiomes de T, de la forme f(x1,.....,xn)=u (u étant un terme externe). Intuitivement une spécification est suffisament complète si les opérateurs externes (observateurs) suffisent pour décrire le comportement du type.

## IV.1.6. LES LANGAGES D'IMPLEMENTATION DE TYPES ABSTRAITS.

La notion de type abstrait de données est apparu dans plusieurs langages et sous différentes formes. Parmi ces langages citons CLU, ADA, SIMULA, ATM. Dans SIMULA le concept de type abstrait est décrit par la notion de CLASSE. Une CLASSE est formé d'un ensemble d'attributs, d'une partie opérations et une partie action (généralement d'initialisation). Dans une CLASSE les parties spécification et implémentation ne sont pas séparées. Dans CLU et ADA ces deux parties sont clairement séparées grâce au concept de CLUSTER pour CLU et PACKAGE pour ADA. Dans le langage ATM, l'accent est mis sur l'abstraction et modularité grace aux concepts de MACHINE, MODULE et CAPSULE décrits séparement.

Outre ces langages, des systèmes d'aide à la spécification tels que OASIS, SPES ont été développés.

## IV.1.7. LA SPECIFICATION COMME OUTIL DE CONCEPTION : INTRODUCTION A LA SPECIFICATION DU SYSTEME SAGE.

La deuxième partie de ce chapitre décrit une étude formelle des concepts utilisés dans la conception du systeme SAGE en utilisant les techniques de spécification algèbrique de types abstraits. Cette méthode a en effet prouvé son applicabilité dans de nombreux domaines, citons par exemple les bases de données [DUF,80] ,[DER,80] ; la sémantique des langages de programmation [PAI,76] , [FIN,80].

On retrace ci-suit les principaux concepts des spécifications formelles et leur mise en oeuvre dans la conception d'outils logiciels.

a. Abstraction : Elle ne permet de garder au stade de la conception que les concepts clés du problème éloignant le plus possible les détails d'implémentation. Le concepteur définit ses objets et opérations comme des opérateurs d'abstraction mathématique. Les seules contraintes qui lui sont infligeés est de rester cohérent (exprimées dans le cas par exemple de la spécification algèbrique par les notions de consistance et de complètude).

b. Paramètrisation et généricité : la technique de

paramètrisation de types abstraits permet de construire de nouveaux types à partir de modèles de types. Dans notre spécification la parmètrisation de type est utilisé pour définir (instancier) les types utilisateurs. Un type (ou certaines opérations) est dit générique s'il peut être utilisé dans d'autres types. Cette technique, associée aux mécanismes de restriction et enrichissement de types abstraits, permet d'une part de construire des types hiérarchiques et s'avère d'autre part un outil de conception assez puissant puisqu'on peut décrire un type comme un sous-type d'un autre.

## IV.2. SPECIFICATION DU SYSTEME SAGE.

### IV.2.1. INTRODUCTION.

L'objet de cette partie de l'étude est d'appliquer les techniques de spécification algèbrique de types abstraits à la conception du système SAGE. On montre ainsi la faisabilité de ce formalisme à un problème aussi complexe que la conception d'un interface Homme-machine. En effet, nous nous proposons de décrire dans les spécifications les fonctionnalités des outils logiciels réalisés pour le développement d'applications interactives. Ces fonctionnalités incluent : les outils de construction d'écrans et l'interaction avec l'utilisateur.

### IV.2.2. PARTICULARITE DU PROBLEME.

Les théories actuelles sur les types abstraits ne permettent de rendre compte facilement des problèmes d'entrées/sorties [SHA,83]. En effet, avant que la valeur d'une variable soit affichée, il faut que sa valeur soit convertie en une forme affichable correspondant à la perception de l'utilisateur. De la même façon la simple saisie d'une valeur demande un dialogue complexe au fur et à mesure que l'utilisateur introduit ses caractères. Ceci est dû principalement au caractère statique des spécifications formelles. On pourra lire à ce propos les propositions décrites dans [GUT,80] pour exprimer la dynamique spécifié à l'aide de routines. Ces routines servent d'interface à l'utilisateur (programme) et sont décrites en utilisant une généralisation des préconditions de DIJKSTRA [DIJ,76].

Concevoir un gestionnaire de dialogue consiste en (cf. chap.II) :

1. définir les différents types utilisateur intervenant dans l'affichage (cf. signification au § III.4).

2. décrire la façon dont ces divers types d'affichage s'agencent entre eux.

3. décrire le lien avec le programme utilisateur.

La spécification d'un gestionnaire de dialogue doit d'autre part maintenir certaines caractèristiques générales que nous exprimons par les invariants suivants :

a. forte liaison avec le programme utilisateur : ceci signifie qu'à chaque instant de l'utilisation, l'écran affiché donne une image fidèle de l'état des variables de dialogue.

b. possibilité de décrire des types de zones d'affichage avec un niveau d'abstraction compatible avec celui des langages de programmation.

c. séparer l'application du processus du dialogue : ceci signifie d'une part une indépendance vis-à-vis du terminal et d'autre part l'indépendance de l'application du "style" (c.à.d l'ensemble des décisions prises par l'utilisateur final pour interagir avec le système) de dialogue.

La suite de ce chapitre décrit une spécification du système SAGE à l'aide de types abstraits. Elle utilise le formalisme de spécification algèbrique décrit dans la première partie de ce chapitre.

Nous pensons qu'une spécification d'un gestionnaire de dialogue doit rendre compte d'au moins deux aspects du système :

- Un premier aspect concerne la description des divers types de zones d'affichage et leur agencement sur l'écran. En particulier, on doit donner la possibilité de définir les différents types d'affichage souhaités par l'utilisateur.

- Un deuxième aspect concerne l'intéraction avec l'utilisateur. Dans cette partie seront spécifiés les divers contrôle allant du bas niveau(ex : on ne peut entrer un caractère dans une zone protégée) au plus complexes (test de validité d'une zone ou de cohérence entre zones).

Dans notre spécification ces deux aspects ont été pris en compte. On montrera, en particulier, comment le deuxième aspect a été résolu en utilisant les concepts d'invariants et de constructeurs de types.

### IV.2.3. TRAVAUX ANTERIEURS.

Plusieurs travaux ont été entrepris ces dernières années pour spécifier le dialogue homme-machine. Mais dans la plupart des cas ces travaux n'offrent pas un niveau d'abstraction élévé et ne prennent souvent en compte qu'un aspect du dialogue (menu ou écran de saisie). Ainsi STUDER dans [STU,84] propose de décrire la sémantique du dialogue (principalement dans le cas de menus) en utilisant la méthode de Vienne. Si cette méthode s'avère assez puissante pour décrire la sémantique du sytème, elle manque néanmoins d'outils de manipulation de types abstraits, qui offrent la possibilité de

décrire des types hiérarchiques permettant d'exprimer, par exemple, qu'un menu est simplement un sous-type du type plus général: écran de dialogue (cf. §IV.1.4). Dans [SHA,83], l'auteur propose de spécifier le dialogue homme-machine en utilisant des règles de composition, l'aspect sémantique n'a été par contre que brièvement discuté.

Dernièrement, des approches formelles ont été introduites pour spécifier le dialogue homme-machine. Dans [SUF,81], l'auteur décrit une spécification progressive d'un éditeur de texte en utilisant un langage axiomatique reposant sur la notion de liste. SUFFRIN décrit la liason entre le document édité et l'écran physique en utilisant une fonction de représentation. Dans [GUT,80], GUTTAG donne une spécification d'un interface homme-machine de haut niveau en utilisant un langage algèbrique. Bien que certains aspects ne sont pas complètement mis en évidence (la définition des types utilisateur et la description de l'intéraction système-utilisateur), ces travaux constituent néanmoins un modèle pour concevoir une spécification et montrer l'interêt d'une spécification comme outil de conception (cf. partie discussion dans le même article).

## IV.2.4. APPROCHES CHOISIES.

La spécification du système se compose de deux parties (cf. § IV.2.2) :

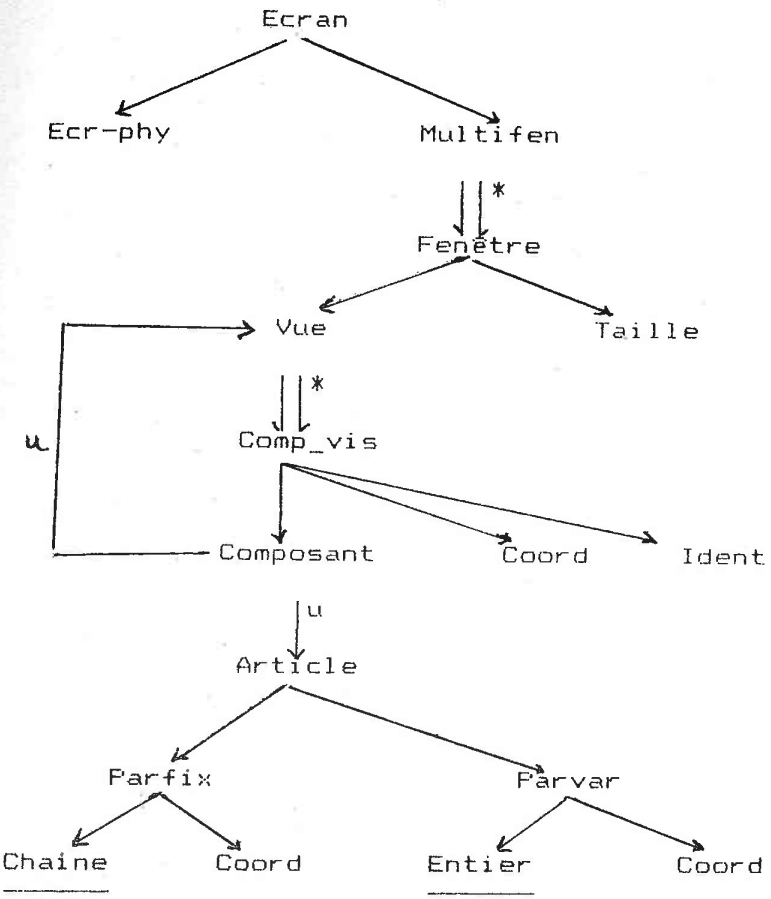
a. Dans la première partie, on décrit les opérations de construction des différentes entités d'affichage en introduisant des opérations et les types appropriés (univers).

b. Dans la deuxième partie, on décrit les opérations de mise en oeuvre des différentes entités sous forme de routines (principalement les opérations d'affichage et de lecture d'un Cadre).

Pour spécifier les différents types du système on a adopté une démarche descendante en utilisant des types hiérarchiques (cf. § IV.1.6). Un des points délicats de la spécification est de décrire la liaison entre les structures d'affichage (l'"écran logique") et l'écran physique. Une première idée qu'on peut avancer est de décrire l'écran physique commen une représentation (au sens fonction de représentation des types abstraits) de l'écran logique. Mais cette idée a l'inconvénient de dissocier écran physique et structures d'affichage alors que les deux types sont présents simultanément. Une meilleure solution, en accord avec la démarche descendante, est de considerer les structures d'affichages comme des sous-types (étendant ou restreignant) de l'écran physique.

Un autre interêt de cette approche est de définir des opérations hiérarchisées utilisables tout le long de la spécification (cf. fonction Appar ci-après) .

La fig. IV.1. donne la liste des différents types intervenant dans la spécification et la relation (hiérarchie) entre eux .




Légende:

/\ : produit cartésien;
/ \

|| * : ensemble

| u : Union

fig. IV.1

## IV.2.5. NOTATIONS.

### a. Désignation des opérations:

Dans la description hiérarchiques des différents types du sytème, la spécification d'un type peut utiliser d'autre types (soit de base, soit spécifiés ultérieurement). Pour distinguer le type qu'on est entrain de spécifier (appelé sorte d'interêt) des autres types on a adopté les notations suivantes:

-La clause sortes indique les différents types qui interviennent dans la spécification du type d'interêt.

-Les opérations des autres types (autre que le type d'intérêt) sont désignés par un nom complet (en utilisant la notation pointée "dot") en précisant en particulier le type propriétaire.

### b. La clause soit (ang. "let"):

Cette clause est utilisée pour définir une fonction ou une constante à partir des autres opérations du type. Son seul interêt est de faciliter l'écriture et la compréhension de la spécification.

### c. Opérations sur les fonctions:

Dans la spécification nous sommes amenés à définir certaines opérations sur les fonctions. Nous préférons donner ci-suit leur forme générale et les utiliser dans les spécifications:

#### i. Restriction:

On note | l'opérateur de restriction d'une fonction à un domaine donné :

soit f : E ——> F et S un sous-ensemble de E

on définit:

f|S : E ——> F par

f|S(x)=f(x) si x ∈ S

*

f|S(x)=vide sinon

ii. Union:

On note θ cet opérateur.
Soient f : E ——> F; g: G ——> F

on définit : fθg : E U G ——> F
*
fθg(x)=si g(x)=vide alors f(x)
sinon g(x)

* : vide est à interpreter indéfini.

d. Commentaires:

Les commentaires suivants sont insérés dans les spécifications:

i.Chaque spécification d'un type est précédée d'une signification informelle du type.

ii. Chaque opération (profil) est precédée d'un symbole désignant son type : (c) : constructeur, (o) : observateur, (s) : simplificateur.
iii. Un commentaire sommaire (commencant par ";") suit la déscription (profil) de chaque opération.

## IV.2.6. SPECIFICATION DES TYPES.

### IV.2.6.1. LE TYPE ECRAN.

L'écran : il représente le document tel qu'il est vu par l'utilisateur final. Il est décrit comme un produit cartésien de l'"écran logique" et l'écran physique. Les invariants inclus dans la spécification de ce type maintiennent les relations entre l'écran physique et l'écran logique (relations qui déterminent l'espace d'affichage et l'emplacement du cureseur). Les opérations définies sur le type écran sont celles qui seront disponibles pour l'utilisateur final. Elles se répartissent entre le niveau fenêtre et le niveau caractère.

type Ecran

sortes Ecr-phy, Multifen, Coord, Car, Bool, Direction, Liste, Ident

opérations

(c) Creer : Ecr-phy X Multifen ——> Ecran
; creer un écran à partir d'un Ecr-phy et une Multifen.

(o) Vide? : Ecran ——> Bool
; écran vide ?

(o) Appar : Ecran X Coord ——> Car
; apparence d'un caractère à une psition donnée.

(o) Curs : Ecran ——> Coord
; position courante du curseur.

(c) Deplc : Ecran X Direction ——> Ecran
; déplacer le curseur d'une position éléméntaire dans une direction donnée.

(c) Lircar : Ecran X Car ——> Ecran
; lecture d'un caractère à la position du curseur.

(s) Effcar : Ecran ——> Ecran
; effacement d'un caractère à la position du curseur.

(o) Fen-act: Ecran ——> Ident
; donne l'identificateur de la fenêtre active.

(c) Deplf : Ecran X Ident X Coord ——> Ecran
; déplacer une fenêtre désignée par un identificateur à une coordionnée donnée.

(s) Supf : Ecran X ident ——> Ecran
; suppression d'une fenêtre désignée par un identificateur.

axiomes :

e : Ecr-phy ; mf : Multifen ; ch : Car ; c : Coord ;
d : Direction ; idf : Ident

(1) Appar(Creer(e,mf))=[Ecr-phy.Appar(e) θ Multifen.Appar(mf)]
| Ecr-phy.Surfe(e)

(2) Vide?(Creer(e,mf))=Multifen.Vide?(mf)

(3) Lircar(Creer(e,mf),ch)=Creer(Ecr-phy.Lircar(e,ch),mf)

(4) Effcar(creer(e,mf))=Creer(Ecr-phy.Effcar(e),mf)

(5) Curs(Creer(e,mf))=Ecr-phy.Curs(e)

(6) Deplc(Creer(e,mf),d)=Creer(Ecr-phy.Deplc(e,d),mf)

(7) Fen-act(Creer(e,mf))=Liste.Tete.Trouvf(mf,Curs(e))

(8) Deplf(Creer(e,mf),idf,c)=Creer(e,Multifen.Deplf(mf,idf,c))

(9) Supf(Cree(e,mf),idf)=Creer(e,Multifen.Supf(mf,idf))

ftype

Explications :

Ces axiomes montrent comment sont obtenues les opérations du type écran à partir de celle de Ecr-phy et Multifen.

### v.2.6.2. LE TYPE ECR-PHY.

écran physique : Il représenta l'abstraction du terminal. Il est défini par sa taille et les opérations de base (curseur, lecture d'un caractère, effacement d'un caractère, déplacement du curseur ...).

ype Ecr-phy

sortes Coord, Taille, Car, Direction

pérations

(c) creer : Taille X Coord ——> Ecr-phy
; créer un écran vide.

(o) Surfe : Ecr-phy ——> Taille
; taille de l'écran.

(o) Curs : Ecr-phy ——> Coord
; position courante du curseur.

(o) Appar : Ecr-phy X Coord ——> Car
; apparence d'un caractère à une coordonnée donnée

(c) Deplc : Ecr-phy X Direction ——> Ecr-phy
; déplacer le curseur d'une position élémentaire dans une direction donnée.

(c) Lircar : Ecr-phy X Car ——> Ecr-phy
;lecture d'un caractère à la position du curseur.

(s) Effcar : Ecr-phy ——> Ecr-phy;
; effacement d'un caractère sous le curseur.

invariants

∀ e:Ecr-phy Curs(e) ∈ Surfe(e)
dom(Appar(e))=Surfe(e)

axiomes

soit

```
    Vdir : Coord;
    Vdir(d) = cas d où
                haut : (0,1)
                bas  :(0,-1)
                droi :  (1,0)
                gau  :  (-1,0)
              fcas

    Vd = Vdir(droi)

 e : Ecr-phy; c,cu : Coord; s : Taille; ch : Car; d : Direction;
(1)  Curs(Ccreer(s,cu)=cu
(2)  Surfe(creer(s,cu))=s
(3)  Appar(creer(s,cu))=Car.vide
(4)  Appar(Deplc(e,d),c)=Appar(e,c)
(5)  Curs(Deplc(e,d))=Coord.Plus(Curs(e),Vdir(d))
(6)  Curs(Lircar(e,ch))=Coord.Plus(Curs(e),Vd)
(7)  Appar(Lircar(e,ch),c)=si Coord.Egal(Curs(e),c)
                           alors ch
                           sinon Appar(e,c)
(8)  Effcar(Lircar(e,ch))=e
```

ftype

Explications:

L'écran physique est vu comme une page de caractères avec des opérations de manipulation (déplacement du curseur, lecture d'un caractère, effacement d'un caractère). Tous ces opérations peuvent être utilisées dans la spécification des autres types sans qu'on ait besoin de les redéfinir. Parmi ces opérations la fonction Appar jour un rôle important dans la spécification. Elle prend une coordonnée et renvoie le caractère se trouvant à cette position s'il existe, le caractère vide sinon. Neanmoins, à chaque fois qu'on l'utilisera dans un type on redéfinira son profil.

L'invariant exprime que le curseur ne peut pas se trouver en dehors des limites de l'écran.

Les axiomes 4 et 5 montrent que le déplacement du curseur n'affecte pas la fonction Appar, par contre le curseur est déplacé d'une position suivant la direction choisie.

Les axiomes 6 et 7 signifient qu'après la lécture d'un

caractère, le curseur est déplacé d'une position à droite et que l'application de la fonction Appar à la position du curseur donne le caractère lu.

L'axiome 8 montre que l'effacement d'un caractère après sa lecture n'a aucun effet sur l'écran .

Remarque:

Les axiomes telles que: Effcar(creer(s,cu)) n'apparaissent pas dans la spécification, ce qui signifie que notre spécification est incomplète. En fait, ces axiomes traduisent généralement des cas d'erreurs. Nous préférons ne pas les décrire directement dans la spécification.

### IV.2.6.3. LE TYPE MULTIFENETRES.

Multifen : C'est un arrangement de l'écran en fenêtres rectangulaires sémantiquement indépendantes. Toutes les dispositions sont permises même celles où deux fenêtres se recouvrent.

type Multifen

sortes Fenêtre,Coord,Ident,Liste,Car,Bool

operations

(c) Creer : () ——> Multifen
; creer une multifen vide.

(c) Ajoutf: Multifen X Fenêtre X Coord X Ident ——> Multifen
; ajouter une fenêtre désignée par un identificateur à une coordonnée donnée

(s) Supf : Multifen X Ident ——> Multifen
; supprimer une fenêtre désignée par son identifiant

(o) Trouvf: Multifen X Coord ——> Liste(Ident)
; trouver une (ou des) fenêtre(s) contenant une coordonnée donnée.

(o) Appar : Multifen X Coord ——> Car
; apparence d'un caractère à une coordonnée donnée.

(o) Vide? : Multifen ——> Bool
; Multifen vide?

axiomes

mf : Multifen ; f : Fenetre ; c,c' : Coord ; idf,idf' : Ident

(1) Vide?(Creer())=vrai

(2) Appar(Creer(),c))=Car.Vide

(3) Trouvf(Creer(),c)=Liste.creer()

(4) Vide?(Ajoutf(mf,f,c,idf))=faux

(5) Appar(Ajoutf(mf,f,c,idf),c')=
si Fenetre.Ed(f,Coord.Rel(c',c))
alors Fenetre.Appar(f,Coord.Rel(c',c))
sinon Appar(mf,c')

(6) Trouvf(Ajoutf(mf,f,c,idf),c')=
si Fentere.Ed(f,Rel(c',c))
alors Liste.inser(Trouvf(mf),idf)
sinon Trouvf(mf,c')

(7) Supf(Ajoutf(mf,f,c,idf),idf')=
si idf=idf'
alors mf
sinon Ajoutf(Supf(mf,idf'),f,c,idf)

type

Explications

Une Multifen est un arrangement de fenêtres où toutes les dispositions sont permises (y compris le recouvrement) . Une Multifen possède toujours une fenêtre active, c'est celle qui a été dernièrement ajoutée.

L'axiome 5 spécifie qu'en cas de recouvrement l'apparition d'un caractère dépend uniquement de la fenêtre active.

Les axiomes 6 et 7 montrent que les fenêtres sont organisées sous forme d'une file d'attente.

### IV.2.6.4. LE TYPE FENETRE.

Fenêtre : C'est la zone d'affichage. Elle est définie par sa taille et la vue qui lui est associée. C'est ce concept qui permet de réaliser l'indépendance entre le terminal et les structures d'affichage.

type Fenetre

sortes Vue,Taille,Coord,Direction,Bool,Car

opérations

(c) Creer : Vue X Taille X Coord ——> Fenetre
; créer une fenêtre

(o) Appar : Fenetre X Coord ——> Car
; apparence d'un caractère à une position donnée.

(o) Ed : Fenetre X Coord ——> Bool
; est-ce qu'une coordonnée donnée se trouve dans la fenêtre?

(s) Scroll: Fenetre X Direction ——> Fenetre
; changer la position de la vue dans la fenêtre.

axiomes

v : Vue ; t : Taille ; c,c' : Coord ; d : Direction

(1) Appar(Creer(v,t,c),c')=Vue.Appar(v,Coord.Rel(c',c))
(2) Ed(Creer(v,t,c),c')=Taille.Ed(Coord.Rel(c',c))
(3) Scroll(Creer(v,t,c),d)=Creer(v,t,c,Coord.Plus(c,Vdir(d)))

ftype

Explications

Une fenêtre est détérminée par la donnée d'une Taille (sa dimension), une Vue et sa position dans la Fenêtre. La modification de la taille permet d'agrandir l'image de la Vue. Les opérations de "Scroll" permettent de visualiser les parties cachées de la Vue.

### IV.2.6.5. LE TYPE VUE .

Vue : C'est un arrangement géographique de composants. Tout recouvrement est interdit.

type Vue (Comp_vis) Comp_vis possède Ed,Egal,Inters,Appar

sortes Comp_vis,Coord,Typdd,Car,Bool

opérations

(c) Creer : () ——> Vue
; Vue vide

(c) Ajoutc: Vue X Comp_vis ——> Vue
; ajouter un composant à une Vue.

(s) Supc : Vue X Comp_vis ——> Vue
; supprimer un composant d'une Vue.

(o) Trouvc: Vue X Coord ——> Comp_vis
; trouver le composant se trouvant à une coordonnée donnée.

(o) Recouv: Vue X Comp_vis X Comp_vis ——> Bool
; est-ce que deux composants se recouvrent?

(o) Appar : Vue X Coord ——> Car
; apparence d'un caractère à une position donnée.

(o) Orig : Vue ——> Coord
; origine (coin en haut à gauche) de la surface délimitée par la Vue.

(o) Cop : Vue ——> Coord
; coin opposé (en bas à droite).

(o) Vide? : Vue ——> Bool
; Est-ce une Vue vide?

définitions

soit

Ed : Vue X Coord ——> Vue
∀ v : Vue ; c : Coord
Ed(v,c)=Coord.entre(Orig(v),Cop(v))

axiomes

v : Vue ; cpv,cpv',cpv" : Comp_vis ; c,c' : Coord

(1) Cop(Creer())=Orig(Creer())
(2) Recouv(Creer(),cpv,cpv')=faux
(3) Appar(Creer(),c)=Car.Vide
(4) Trouvc(Creer(),c)=Comp_vis.Vide
(5) Vide?(Creer())=vrai
(6) Vide?(Ajoutc(v,cpv))=faux
(7) Orig(Ajoutc(v,cpv))=Coord.Min(Orig(v),Orig(cpv))
(8) Cop(Ajoutc(v,cpv))=Coord.Max(Cop(v),Cop(cpv))
(9) Trouvc(Ajoutc(v,cpv),c)=si Comp_vis.Ed(cpv,c)
alors cpv
sinon Trouvc(v,c)
(10) Appar(Ajoutc(v,cpv),c)=si Comp_vis.Ed(cpv,c)
alors Comp_vis.Appar(cpv,c)
sinon Appar(v,c)
(11) Recouv(Ajoutc(v,cpv),cpv',cpv")=
si Comp_vis.Egal(cpv,cpv')
alors ¬Vide?(Comp_vis.Inters(cpv,cpv"))
sinon si Comp_vis.Egal(cpv,cpv")
alors ¬Vide?(Comp_vis.Inters(cpv,cpv'))
sinon Recouv(v,cpv',cpv")
(12) Supc(Ajoutc(v,cpv),cpv')=si Comp_vis.Egal(cpv,cpv')
alors v
sinon Ajoutc(Supc(v,cpv'),cpv)

préconditions

pre Ajoutc(v,cpv)=non(v=Ajoutc(v',cpv') et Recouv(v,cpv,cpv'))

ftype

Explications

Le type Vue est un type central dans la spécification. Il permet de définir les différents types utilisateurs (Sous-Cadre, Menu, Tableau ...). Il est paramètré par le type Composant (par l'intérmediaire du type Comp_vis). Tout Composant peut être un paramètre de Vue s'il possède les opérations Egal, Appar, Ed et Inters.

Les axiomes.

Contrairement au type Fenetre, une Vue n'a pas de taille prédéterminée. Elle dépend des tailles et de la disposition de ses différents composants. Plus précisement, la taille d'une Vue est déterminée par le rectangle de dimension minimale englobant tous les composants. C'est pour pouvoir faire ces calculs qu'ont été introduites les fonctions Orig et Cop.

Les axiomes 1 2 3 4 donnent les valeurs de Orig, Cop, Appar et Recouv pour une Vue vide.
Les axiomes 7 et 8 calculent les valeurs de Orig et Cop après adjonction d'un Composant, ils utilisent les fonctions les fonctions Min et Max définies sur le type Coord(cf. § IV.2.6).
Les axiomes 9 10 11 12 décrivent les relations classiques du type ensemble.

## IV.2.6.6. LE TYPE COMPOSANT VISUALISE.

type Comp_vis (Composant)

Compsant possède Appar,Lim

sortes Composant,Coord,Car,Ident,Bool

opérations

(c) Creer : Composant X Coord X Ident ——> Comp_vis
; créer un Compo_vis à partir d'un composant, une coordonnée et identificateur.

(o) Orig : Comp_vis ——> Coord
; origine (coin haut à gauche) de la surface délimitée par le composant.

(o) Cop : Comp_vis ——> Coor
; coin opposé (en bas à droite).

(o) Appar : Comp_vis X Coord ——> Car
; apparence d'un caractère à une position donnée.

(o) Ed : Comp_vis X Coord ——> Bool
; une coordonnée se trouve-t-elle dans un composant?

(c) Inters: Comp_vis X Comp_vis ——> Comp_vis
; Compo_vis obtenu par intersection de deux Comp_vis

(o) Vide? : Comp_vis ——> Bool
; Est-ce le Comp_vis vide?

axiomes

```
  cpv : Comp_vis ; cp,cp' : Composant ; c,c' : Coord ;
  idc,idc': Ident

(1) Orig(Creer(cp,c,idc))=c
(2) Cop(Creer(cp,c,idc))=Coord.Plus(c,(Lim(cp).larg,Lim(cp).Haut))
(3) Appar(Creer(cp,c,idc),c') =
        si Ed(Creer(cp,c,idc),c')
        alors Composant.Appar(cp,Coord.rel(c',c))
        sinon Car.Vide
(4)' Ed(cpv,c)=Coord.Entre(Orig(cpv),Cop(cpv))
(5) Egal(Creer(cp,c,idc),Creer(cp',c',idc'))=(idc=idc')
(6) Orig(Inters(cpv,cpv')) =
      si Ed(cpv',Orig(cpv))
      alors (Orig(cpv).Lig,Orig(cpv').Col))
      sinon si Ed(cpv',(Orig(cpv).Lig,Cop(cpv).Col))
            alors (Orig(cpv).Lig,Orig(cpv').Col))
            sinon si ¬(Ed(cpv',Orig(cpv)) et
                     ¬(Ed(cpv',(Orig(cpv).Lig,Cop(cpv).Col))
                  alors Coord.Vide
                  sinon Orig(Inters(cpv',cpv))
(7) Cop(Inters(cpv,cpv'))=idem
(8) Appar(Inters(cpv,cpv'),c)=
      si Ed((Inters(cpv,cpv'),c)
      alors choix(Appar(cpv,c),Appar(cpv',c))
      sinon Car.Vide
(9) Vide?(cpv)=Coord.Egal(Orig(cpv),Cop(cpv))
```

ftype

Explications

Le type Comp_vis (Composant visualisé) est un type intermediaire qui a été introduit simplement pour faciliter la spécification du type Vue.Un Comp_vis est déterminé par la donnée d'un Composant, une coordonnée (sa position) et un identifiant.
Les axiomes 1 et 2 donnent les valeurs de Orig et Cop en fonction de la taille du Composant .
L'axiome 3 spécifie comment est déterminé Ed (Est_dans) en fonction des attributs Orig et Cop.
L'axiome 4 lie Appar d'un Comp_vis à celle d'un Composant.
L'axiome 5 spécifie que deux Copm_vis sont égaux si leur identifiant sont égaux.
Les axiomes 6 et 7 déterminent les attributs Orig et Cop de la zone d'intersection. Ces attributs sont calculés en énumérant tous les cas possibles de recouvrement.
L'axiome 8 exprime que l'apparence dans la zone d'intersection est une fonction choix entre l'apparence des deux composants en recouvrement.
L'axiome 9 spécifie dans quel cas un Comp_vis est vide.

## IV.2.6.7. LE TYPE COMPOSANT.

Composant : Un composant est soit un Article, soit une Vue. C'est cette récursivité entre Composant et Vue qui permet de définir les différents types utilisateurs : Sous-Cadre, Menu, Tableau ...etc.

type Composant

sortes Vue, Article, Taille, COOrd, Car

opérations

(c) Art_comp : Article ——> Composant
; Article en tant que Composant.

(c) Vue_comp : Vue ——> Composant
; Vue en tant que Composant.

(o) Appar : Composant X Coord ——> Car
; apparence d'un caractère à une position donnée.

(o) Lim : Composant ——> Taille
; taille de la surface délimitée par le composant

axiomes

Composant = Article U Composant

ftype

Explications

Composant est simplement l'union des types Article et Vue.

## IV.2.6.7. LE TYPE ARTICLE.

Article : Un article est formé d'une partie fixe (zone protégée) et une partie variable (zone non protégée).

type Article

sortes Chaine,Entier,Coord,Taille,Bool

opérations

(c) Creer : Chaine X Entier X Coord X Coord ——> Article
; créer un Article

(o) Posfix: Article ——> Coord
; position de la partie fixe

(o) Posvar: Article ——> Coord
; position de la partie variable

(o) Appar : Article X Coord ——> Car
; apparence d'un caractère à une position donnée.

(o) Lim : Article ——> Taille
; taille de la surface délimitée par un Article.

(o) Fvide?: Article ——> Bool
; Est-ce que la partie fixe est vide?

(o) Vvide?: Article ——> Bool
; Est-ce que la partie variable est vide?

axiomes

pf : Chaine ; pv : Entier ; cf,cv,c : Coord

(1) Posfix(Creer(pf,pv,cf,cv))=cf
(2) Posvar(Creer(pf,pv,cf,cv))=cv
(3) Appar(Creer(pf,pv,cf,cv))=si Coord.Entre(cf,c,cf')
alors pf(c)
sinon Car.vide
(4) Lim(Creer(pf,pv,cf,cv)).Haut=
Entier.Max(Lig(cf)+pv,Lig(cv)+Long(pf)) -
Entier.Min(Lig(cf)+Long(pf),Lig(cv)+pv)
(5) Lim(Creer(pf,pv,cf,cv)).Larg=idem.
(6) Fvide?(Creer(pf,pv,cf,cv))=Chaine.Vide?(pf)
(7) Vvide?(Creer(pf,pv,cf,cv))=(pv=0)

ftype

Explications

Un article est formé d'une partie fixe (zone protégée) de type chaîne (éventuellement vide) et d'une partie variable (zone non protégée) spécifié par sa longueur et leur position respective.
L'axiome 3 détermine l'apparence d'un Article à une coordonée donnée.
L'axiome 4 détermine la taille d'un article en fonction des paramètres de création .

## IV.2.7. LES TYPES UTILITAIRES.

### IV.2.7.1. LE TYPE LISTE.

type Liste(Val)

opérations

```
Creer : ()              ---> Liste
        ; liste vide

inser : Liste X Val     ---> Liste
        ; insérer un élément dans la liste

Long  : Liste           ---> Entier
        ; longueur de la liste

Vide? : Liste           ---> Bool
        ; est-ce que la liste est vide?
```

axiomes

l : Liste ; v: Val

(1) Vide?(Creer)=vrai

(2) Long(Creer)=0

(3) Vide?(inser(l,v))=faux

(4) Long(inser(l,v))=Long(v)+1

ftype

Explications

Ce sont les axiomes standards du type Liste

## IV.2.7.2. LE TYPE COORD.

type Coord stype Entier X Entier

opérations

```
Lig   : Coord                          ----> Entier
        ; abscisse

Col   : Coord                          ----> Entier
        ; ordonnée

Plus  : Coord X Coord                  ----> Coord
        ; coordonnée obtenue en faisant la somme ligne à
          ligne et la somme colonne à colonne.

Rel   : Coord X Coord                  ----> Coord
        ; coordonnée obtenue en faisant la différence ligne
          à ligne et la différence colonne à colonne.

Entre : Coord X Coord                  ----> Bool
        ; est-ce qu'une coordonnée se trouve entre deux
          coordonnées pour la relation d'ordre
          lexicographique sur l'abscisse et l'ordonnée (inf)
          inf(c,c')=[lig(c) < lig(c')] ou [(lig(c) = lig(c'))
                                           et (col(c) < col(c'))]
Min   : Coord X Coord                  ----> Coord
        ; minimum de deux coordonnées pour la relation "inf"

Max   : Coord X Coord                  ----> Coord
        ; maximum de deux coordonnées pour la relation "inf"
```

axiomes

c,c',c" : Coord

(1) Lig(Plus(c,c'))=Lig(c)+Lig(c')
(2) Col(Plus(c,c'))=Col(c,c')
(3) Lig(Rel(c,c'))=Lig(c)-Lig(c')
(4) Col(Rel(c,c'))=Col(c)-Col(c')
(5) Entre(c,c',c")=[(Lig(c') < Lig(c)) ou
((Lig(c)=Lig(c') et (Col(c) < Col(c'))]
et [(Lig(c) < Lig(c") ou
((Lig(c)=Lig(c") et (Col(c) < Col(c"))]
(6) Lig(Min(c,c'))=Entier.Min(Lig(c),Lig(c'))
(7) Col(Min(c,c'))=Entier.Min(Col(c),Col(c'))
(8) Lig(Max(c,c'))=Entier.Max(Lig(c),Lig(c'))
(9) Col(Max(c,c'))=Entier.Max(Col(c),Col(c'))

ftype

Explications

Le type Coord représente les coordonnées caratésiennes et est décrit comme un sous-type du produit cartèsien Entier X Entier.

IV.2.7.3. LES AUTRES TYPES.

type Chaine = Liste(Car)

type Direcion =(Haut,Bas,Droite,Gauche)

type Taille = Struct (Haut:Entier ; Larg:Entier)

## IV.2.8. DEFINITION DES TYPES UTILISATEUR ET LES OPERATIONS DE MISE EN OEUVRE.

### IV.2.7.1. LES TYPES UTILISATEUR.

Pour définir les types utilisateurs (Sous-Cadre,Menu,Tableau) on utilise le mécanisme d'instantiation des types paramètrés en ajoutant éventuellement des restrictions qu'on exprime à l'aide de préconditions sur les opérations du type paramètré.

a. Le Scadre.

```
type Scadre : Vue (Article)
```

Il n'y a aucune restriction à faire sur ce type.

b. Le Menu.

```
type Menu stype Scadre
restriction
  soit   m,m':Menu ; at,at':Article ; c,c': Coord ; id,id':Ident

pré Ajoutat(m,at,c,id) = non (m=Ajoutat(m',at',c',id') et
                              Fvide?(at) et  Fvide?(at'))
```

Cette précondition permet d'expliciter la description intuitive d'un Menu de type options-choix. Un menu est un ensemble de zones (options) dont au plus une est non protégée (celle servant à recueillir la réponse de l'utilisateur).

c. Le type Tableau.

```
type Tabart(Align) stype Scadre

restrictions

 Soit

    Align =(Horiz,Vert)
    tab,tab':Tabart ; c,c':Coord  ; a:Align ; id,id':Ident

 pre Ajoutat(tab,at,c,id) =

    (a=Horiz) ==> [tab=Ajoutat(tab',at',c',id') et
                   Fvide?(at) et Lig(c)=Lig(c')]
    (a=Vert) ==>  [tab=Ajoutat(tab',at',c',id') et
                   Fvide?(at) et Col(c')=Col(c)]

extension

   Occur : Tabart                  ---> Entier

   Occur(Tabvide) = 0
   Occur(Ajoutat(tab,at,c,id))=Occur(tab)+1
ftype
```

Explications

Cette précondition exprime qu'un tableau est formé d'une zone protégée et d'une suite de zones non protégées de même type et disposées suivant l'alignement donné . L'opération d'extension permet de connaitre le nombre d'occurence d'apparition d'un Article dans un Tableau.

d. Le type Cadre.

```
type Cadre = Vue (Article U Scadre U Menu Tabart)
```

## IV.2.7.2. DESCRIPTION DES ROUTINES.

Il s'agit de définir les opérations de mise en oeuvre d'un Cadre : "aff-cad" et "lec-cad".

### 1. La fonction "aff-cad".

La fonction "aff-cad" prend un nom de Cadre et la bibliothèque associée et fournit un écran en affichage. Son profil est donc :

aff-cad : Bibcad X Identc ——> Ecran

Elle fait intervenir le type Bibcad dont on donne çi-suit la spécification (par soucis de généralité on utilise un type paramètré par le paramètre "entité" ) :

type Bib (Entité)

sortes Bool, Ident

opérations

creer : () ——> Bib
; Bibliothèque vide

ajte : Bib X Entite X Ident ——> Bib
; ajouter une entité désignée par un identifiant

supe : Bib X Ident ——> Bib
; supprimer une entité désignée par son identifiant

acce : Bib X Ident ——> Entite
; acceder à une entité à partir de son identifiant

app : Bib X Ident ——> Bool
; Est-ce qu'une entité se trouve dans la bibliothèque

axiomes

b : Bib ; e : Entite ; ide,ide' : Ident

(1) app(creer(),ide)=faux

(2) app(ajte(b,e,ide),ide')=si ide=ide'
alors vrai
sinon app(b,ide')

(3) supe(ajte(b,e,ide),ide')=si ide=ide'
alors b
sinon ajte(supe(b,ide'),e,ide)

(4) acce(ajte(b,e,ide),ide')=si ide=ide'
alors e
sinon acce(b,ide')

{ acce(creer(),ide) non spécifié }

ftype

Explications

Ce sont les axiomes du type ensemble (plus precisement multi-ensemble).

L'axiome définissant l'opérations "aff-cad" s'écrit alors :

soit bc : Bib(Cadre) ; nc : Ident ; e: Ecr-phy

aff-cad(bc,nc)=Ecran.creer(acce(bc,nc),e)

2. La fonction lec-cad.

Elle a pour profil :

lec-cad : Vue ——> Typdd .

Cette fonction prend un Cadre (ou plus généralement une Vue) en affichage et produit un objet de type Typdd décrivant la représentation interne de la Vue.

En effet, à toute Vue est associé un type donnant la représentation interne des structures d'affichage : Typdd (type

de données). Ce type est soit un type de base : entier, réel ou chaine soit un type obtenu à l'aide de constructeurs de types : intervalle, produit cartésien, tableau, ensemble ...etc ; soit enfin un type obtenu en combinant constructeurs de types et prédicats. Ainsi, par exemple, on pourrait définir le type Date par :

type Date : Struct( J : 1..31 ; M : 1..12 ; A : Entier)

avec

(A > 0)
et (M pair ==> (J ∈ [1..30]) et (M impair ==> (J ∈ [1..31]))

et (M=2 ==> (J ∈ [1..29]))

En utilisant ce procédé on obtient une bibliothèque de types qui peut être mise à jour par l'utilisateur.

Tout Typdd doit posséder une fonction de construction de profil:

interp : (Entier X Entier ——> Car) ——> Typdd

Intuitivement, la fonction Interp génère une valeur du type Typdd à partir d'une fonction (ou plus concrètement d'un tableau à deux dimensions de caractères) :

Ainsi, pour les types de base (on utilisera simplement une fonction à une dimension) :
Chaine :

Interp : (Ent ——> Car) ——> Chaine
la fonction Interp est dans ce cas évidente.

Entier :

Interp : (Ent ——> Car) ——> Entier
Il s'agit dans ce cas d'une conversion chaine ——> entier .

Réel :

Inter : (Ent ——> Car) ——> Reel
idem. entier .

Pour les types structurés, la fonction Interp est obtenue par composition de la fonction de chaque composant.

L'axiome définissant lec-cad s'écrit alors :

soit v : Vue

lec-cad (v) = Typdd.Interp(Vue.Appar(v))

Cet invariant exprime la relation entre l'image affichée et la représentation interne d'un Cadre en liant appar et interp.

# CHAP. V. ARCHITECTURE DU SYSTEME.

## .1. INTRODUCTION.

Le système est composé de quatres modules principaux dont la
ig.V.1 en donne le schéma général et les relations entre eux.
es modules communiquent entre eux en échangeant des variables ou
es procédures selon le mécanisme classique de compilation
éparée. La suite de ce chapitre décrit le rôle de chacun de ces
odules à savoir le Pilote, le Bibliotècaire, le Constructeur,
'Interprète et le module Utilitaires.

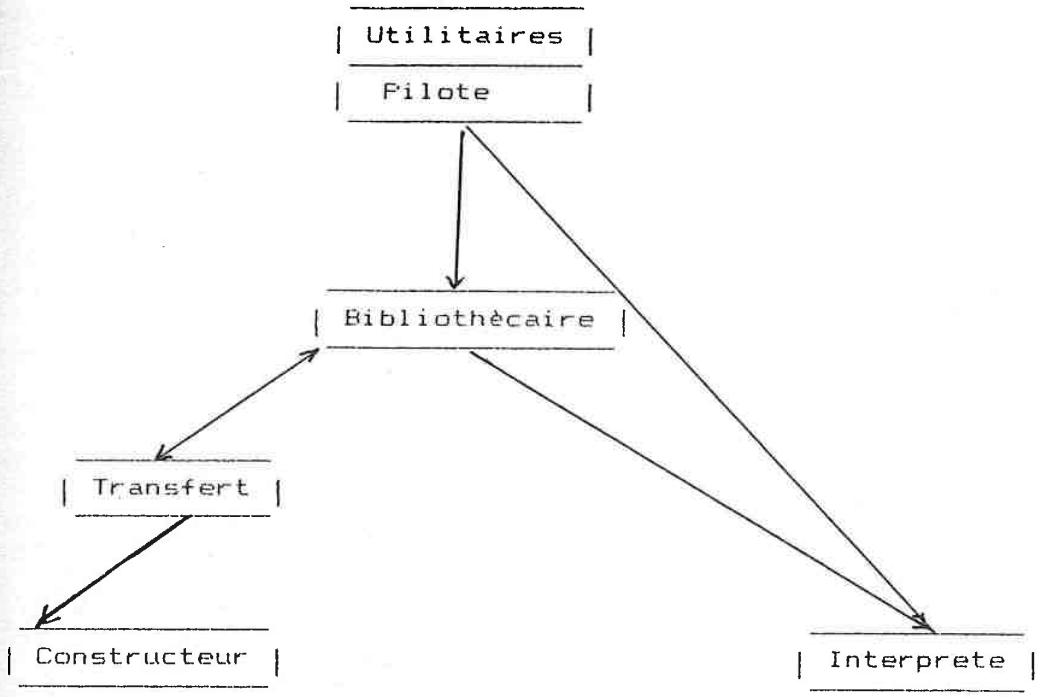



fig. V.1

## V.2. Le Pilote.

SAGE étant lui même un programme interactif, le dialogue avec l'utilisateur est d'une première importance. La gestion du dialogue est confiée au Pilote. En effet le rôle du Pilote est d'enregistrer tous les ordres entrés par l'utilisateur et de l'aiguiller vers un des modules en fonction de la commande choisie. L'utililisateur choisit ses commandes à partir de menus de sélection (construits en utilisant SAGE cf. chap.III § III.5).

L'algorithme de description du dialogue se compose de deux parties (cf. § II.2.4) :

- La première partie concerne l'enchaînement dans le graphe de la fig. III.4 :

```
proc. Enchainement
debut
   etat:=etat_initial;
   repeter
     Trait(etat,rep)
     etat:=Transition(etat,rep)
   jusqu'à etat=etat_final
fproc;
```

- Trait(etat) traite un noeud du graphe (appelé etat) :

```
proc. Trait(etat,rep)
debut
  afficher(ecran);
  repeter
    rep:=saisie(ecran);
    erreur:=action(rep);
    si erreur alors message(rep,erreur)
  jusqu'à non(erreur)
fproc;
```

Trait(etat) consiste à afficher l'écran associé à cet état, saisir la réponse de l'utilisateur et exécuter une action en fonction de la réponse donnée. Message affiche un message d'erreur en cas de réponse erronée.

Pour prendre en compte les touches de fonctions dans les procédures d'édition, le programme s'appuie sur une procédure "lecar" (lecture d'un caractère). La procédure "lecar" enregistre tous les ordres émanant soit de l'écran "programmeur" soit de l'écran "utilisateur" (cf. §III.5.1), exécute certaines fonctions de base (déplacement du curseur, mémorisation dans un tampon,...).

Rappelons que pour un objectif de portabilité, toutes ces fonctions ainsi que d'autres plus complexes (effacement de caractères, retour à la ligne, mode insertion, mode page, effacement de ligne..) ont été "programmées". La procédure "lecar" utilise deux sous-programmes, très simples, écrits en assembleur (de test et lecture directe sur un port d'entrées/sorties) dépendant du système et dont la réécriture pour un autre système ne poserait aucun problème.

Le corps de la procédure "lecar" s'écrit:

```
proc. Lecar;
var
  Port1,Port2      : boolean;
  Pprog,Putil      : Port;          (* ports d'entrées/sorties *)
  Carprog,Carutil  : char;
debut
  répéter
        (* test si une port est prêt *)
    Port1:=Test_port(Pprog);
    Port2:=Test_port(Putil);
  jusqu'à Port1 ou Port2;
  si Port1                          (* port programmeur   *)
  alors
    Carprog:=Lec_port(Pprog);       (* lecture du caractère *)
    cas Carprog où
    "->" : versd(Pprog);
    "<-" : versg(Pprog);
    "^"  : versh(Pprog);            (* déplacement du curseur *)
    "v"  : versb(Pprog);
     RC  : ligsuiv(Pprog);          (* ligne suivante         *)
     DEL : effcar(Pprog);           (* effacement caractère   *)
     INS : inser(Pprog);            (* mode insertion         *)
    fcas
  fsi
  sinon si Port2                    (* port utilisateur       *)
        alors
          ..................
          ..................
             (* idem Pprog *)
          ..................
           CTRL-H : deplace..
           CTRL-S : enleve..
        fsi
fproc;
```

La structuration des différents modules a été faite de telle sorte que les modules sont décrits indépendament. En particulier, une version "ligne-à-ligne" du module Pilote (mode

commandes) existe toujours et peut à tout moment remplacer la version "menus" sans aucun changement des autres modules. La version "menus" a l'avantage, d'une part de faciliter l'utilisation du système et d'autre part de mieux structurer le programme du dialogue.

## V.3. Le BIBLIOTHECAIRE.

Ce module regroupe les opérations d'archivage et de mise à jour des différentes entités. Il s'appuie sur une structuration des différentes entités selon le modèle entité-association dont la fig.V.2 donne le schéma (par soucis de clarté l'entité Tabart a été omise) :

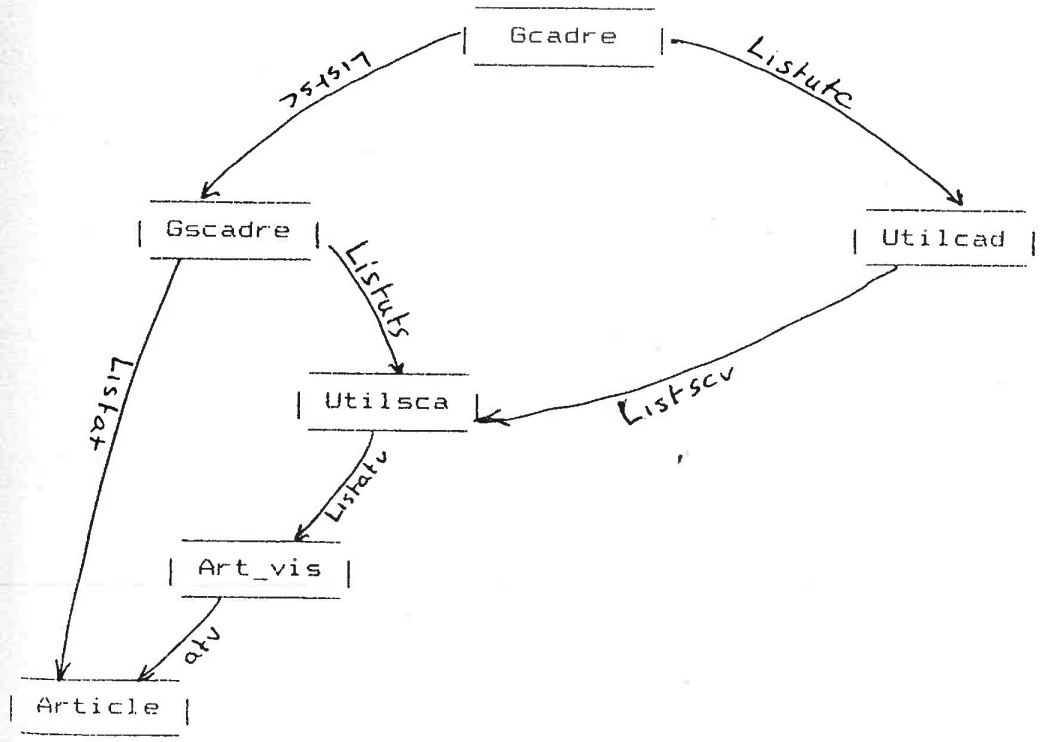



fig. V.2

Le schéma présenté ici n'est qu'une forme simplifiée du Bibliothècaire. En effet, ni l'archivage des contrôles ni les caractèristiques du terminal n'ont été spécifiés. Néanmoins ce module devient vite complexe et constitue une véritable base de données principalement au niveau des cohérences. En effet, outre

l'implémentation des opérations de mise à jour de chaque entité, on doit décrire les relations entre elles.

En effet l'utilisateur souhaiterait, par exemple, connaître pour un Article donné la liste des Scadres où cet Article est utilisé. Ces relations expriment aussi des contraintes au niveau de certaines opérations : par exemple, on ne peut modifier ou supprimer un Article qui est utilisé dans au moins un Scadre ou un Cadre. Par contre la suppression d'une entité n'implique pas la suppression des entités inférieures (qui la constituent).

L'implantation de cette base est faite en utilisant des fichiers à accès directs. Plus exactement on a utilisé des fichiers à accès aléatoire associés à des tables d'index (implémentés sous forme d'arbre de recherche ou arbre binaire). Ceci nous a permis d'une part de faire une gestion des places libres des fichiers (à l'aide d'une pile des places libres) et d'autre part de répondre facilement à des questions telles que : "Liste des éléments d'une bibliothèque".

Les entités utilisées dans la description du schéma de la B.D se divisent en trois sous-ensembles :

i.Les entités élémentaires :
- Article
- Tabart

définies par leurs attributs.

ii.Les entités de type groupe :
- Groupe Scadre
- Groupe Cadre.

définis par la liste des composants qui les constituent. Cette liste est organisée sous forme d'une liste chaînée avec des pointeurs qui pointent vers l'adresse de chaque composant.

iii. enfin les Utilisations :
- Art_vc
- Util_sca
- Util_cad

qui utilisent une structure intermédiaire prenant en compte les caractéristiques d'affichage.

Ainsi on définira par exemple :

```
type
Art_vc = record
           Nat : indice ; (* num. dee l'article dans
                             le groupe *)
           Posfix : Coord;(* position de la partie fixe      *)
           Posvar : Coord;(* position de la partie variable  *)
           Atv    : Attr_vis; (* attributs visuels           *)
         end;
```

```
Utilsca = record
            Nu : indice ; (* num. de l'utilisation          *)
            Listatv : array [indice] of Art_vc; (* Liste des
                      article avec les attributs d'affichage *)
          end;
```

Le module Bibliothècaire est constitué donc en fait de trois sous-modules:
- -Module de gestion de l'arbre.
- -Module de gestion des places libres.
- -Module de gestion des bibliothèques.

## V.4. LE CONSTRUCTEUR.

Le Constructeur regroupe les procédures réalisant les différentes opérations d'édition d'un écran. Son rôle est d'assister l'utilisateur à définir les différentes entités.

La réalisation de ce module est une mise en oeuvre directe des spécifications (cf. chap.IV.2). La première étape de conception consiste d'abord à définir les structures de données adaptées, implémentant les types abstraits décrits dans la spécification. Pour ce faire on a pris la représentation minimale, c.à.d la représentation qui permet d'extraire le maximum suffisant d'informations sur le type en partant d'une spécification suffisament complète. Ainsi dans le cas d'un type défini comme un produit cartésien, on doit représenter au moins toutes les opérations de projection. Dans le cas d'un type "structuré" on doit implémenter les opérations d'"observation".

Appliquant ces idées à la définition des structures de données intervenant dans ce module, on obtient les descriptions suivantes en utilisant les constructeurs de base de PASCAL.

```
type
  Refart = ^Artcons;
  Reftab = ^Tabcons;     (* pointeur sur chaque composant    *)
  Refvue = ^Vue;

     (* Article en construction *)

  Artcons = record
              Noat : string[10];
              Parfix,Parvar : string[80];
              Posfix,Posvar : Coord
            end;
```

```
    (* Tableau en construction  *)

 Tabcons = record
           -------
             Ebas : Artcons;
             Nboc : integer;
             Alig : (HORIZ,VERT);
             Esp  : integer
           end;
           ---

    (* Un composant quelconque  *)

  Composant = record
              --------
               case Typc : (ART,SCA,TAB) of
               -----                     --
                ART : (Art_comp : Refart);
                SCA : (Sca_comp : Refvue);
                TAB : (Tab_comp : Reftab)
               end
               ---
              end;
              ---

    (* Un composant visualisé    *)

 Comp_vis = record
            -------
              Comp : Composant;
              Orig,Cop : Coord
            end;
            ---

      (* Une Vue : ensemble de Composants *)

 Vue = record
       -------
         Nov : string[10];
         Listc : array [indice] of Comp_vis;
         Dim   : record Haut,Larg : integer end;
                 ---------                ---

       end;
       ---
```

Pour implémenter la récursivité entre Vue et Composant signalée dans la spécification (cf. § IV.2.4), on utilise ici la possibilté de définir en PASCAL des structures de données récursives.

Conformément aux spécifications, les procédures suivantes ont été implémentées :

```
procedure cre_comp(cp:Composant;pos:Coord;var cpv:Comp_vis);
     (* cette procédure crée un composant visualisé à partir
     d'un composant et ses coordonnées                        *)

procedure inter_comp(cpv,cpv':Comp_vis;var z:Zone);
     (*  détermine la zone  d'intersection (origine et coin
         opposé) de deux composants visualisés  *)

procedure cop_comp(var v:Vue;cpv:Comp_vis);
     (* insère un composant dans une Vue         *)

procedure sup_comp(var V:Vue;idc:Indice);
     (* supprime un composant d'une Vue  *)

procedure aff_comp(var cpv:Comp_vis);
     (* affiche un composant                     *)

procedure depl_comp(var V:Vue;idc:Indice;pos:Coord);
     (* déplace un composant                *)

procedure trouv_comp(var V:Vue;c:Coord;var idc:Indice);
     (* trouve le composant se trouvant à la coordonnée c *)
```

Le Constructeur ne travaille qu'en mémoire centrale en maintenant les différentes structures décrites précedement, qu'on pourrait appeler représentaion externe par opposition à la représentation interne dans la bibliothèque.

Le passage de la représentation interne à la représentation externe et vice-versa se fait par un module spécial : le module de Transfert (voir fig.). Le rôle de ce module est faire les "conversions" entre représentation interne et représentation externe pour chaque type d'entité.

## V.5. LE MODULE UTILITAIRES.

Ce module sert de "boite à outils" pour les autres modules. Dans ce module sont regroupées les fonctions suivantes:

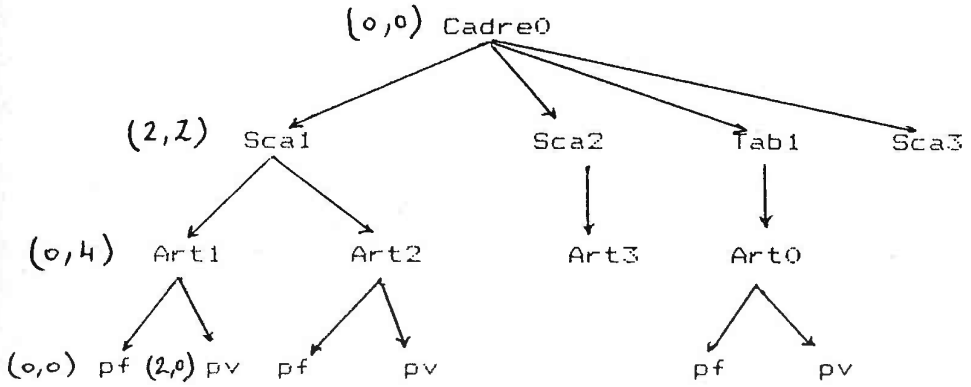
- Suivi du curseur.
- Trace papier : cette fonction permet d'avoir, sur l'imprimante connectée à l'ordinateur, une image du cadre en création.
- Aides : * liste des commandes disponibles
  * liste des éléments d'une bibliothèque.

- Gestion des messages : * commentaires
  * messages d'erreurs.
- Initialisation et modification des paramètres du système :
* taille de l'écran
* caractèristiques vidéo du terminal
* équivalences ordres <—> touches de fonctions.
- utilitaires divers :
  * conversions minuscules <—> majuscules
  * conversions entier <—> chaine

etc...

## V.6. L'INTERPRETE.

Ce module regroupe les procédures qui permettent d'exploiter les écrans dans le programme utilisateur. Il s'agit d'une interprétation incrèmentale des descriptions crées à l'aide du constructeur. En effet, la description d'un Cadre permet de définir une structure arborescente de la forme :




En particulier, la position finale de la partie variable d'un Article est obtenue par composition (on parle dans ce cas d'attribut synthètisé) des positions des noeuds supérieurs. Ainsi dans l'exemple précédent la position finale de la partie variable de l'Article Art1 du Sous-Cadre SCA1 est (4,6) si la position de l'origine du Cadre est (0,0).

Ce mécanisme d'interprétation donne le "squelette" de toute procédure portant sur un Cadre pour exécuter une action ou définir une propriété donnée :

```
procedure action(var cd:Cadre);
debut
  avec cd faire
    pour chaque composant faire
      cas composant où
        SCA : pour chaque article faire act_art(article);
        TAB : pour n=1 à Nboc faire act_art(art_de_base);
        ART : act_art(art_comp)
      fcas
    fpour
  favec
fproc;
```

Parmi ces procédures deux servent d'interface avec le programme utilisateur : "aff_ecr" et "lec_ecr".

1)-La procédure "aff_ecr" affiche un écran désigné par son nom et le numéro d'utilisation voulue ; initialise toutes les parties variables à vide si c'est le premier affichage de cet écran, aux dernières valeurs si non. Le corps de la procédure "aff-ecr" s'exprime donc par :

```
procedure aff-ecr(noecr:string;util:integer);
var
    Premier_appel : boolean;
    cd            : Cadre;
debut
  si Premier_appel[noecr]
  alors
     Premier_appel[,oecr]:=faux;
     cd:=accès_bib(noecr,util);
     avec cd faire
       pour chaque composant faire
        cas composant où
         SCA : pour chaque article faire aff_art(article);
         TAB : pour n=1 à Nboc faire aff_art(art_de_base);
         ART : aff_art(art-comp);
        fcas;
        init_zonzs;
       fpour
     favec
  fsi
fproc;
```

"aff-art" affiche les parties fixe et variable d'un Article après détermination de leur positions finales.

2)-La procédure "lec_ecr" saisit les zones variables du Cadre en affichage, exécute les contrôles spécifiés lors de la construction du Cadre et produit un tampon formaté directement exploitable par le programme utilisateur (en garnissant la valeur d'un variable du type généré à la construction (cf. chap. III § III.4.4).

L'exécution des contrôles se compose, en fait, de deux parties (cf. chap.III § III.4.3.2 ) :

i. la première partie concerne les contrôles statiques liés au format d'affichage. Dans cette partie, on se limite à vérifier que les caractères entrés par l'utilisateur correspondent au format d'affichage donné lors de la construction de l'Article. La

non correspondance est signalée à l'utilisateur par le bloquage du clavier et le déclenchement d'une sonnerie.

ii. la deuxième partie concerne les contrôles dynamiques liés au test de cohérence entre valeurs des différentes zones du Cadre. Il s'agit dans cette partie comme dans les cas précédents d'une interprètation incrémentale de la structure du Cadre. En effet, comme il a été expliqué au § III.4.3.2 la description des contrôles associés à un Cadre comprend tous les contrôles liés à chacun des composants et ceux liés à leur regroupement dans le Cadre.

La procédure d'interprétation s'écrit donc (cf. § IV.2. ):

```
procedure interp(str:string;var cd:cadre);
  debut
    avec cd faire
      pour chaque composant faire interp_comp(str,composant);
      interp_contrôles
    favec
fproc;
```

Enfin la procédure "lec-ecr" s'écrira :

```
procedure lec-ecr(var cd:Cadre);

var
  str      : string[255];
  err      : boolean;
  car      : char;

debut
  repeter
    car:=lecar;
    err:=verif(format,car);
    str:=concat(str,car)
  jusqu'à non (err)
   interp(str,cd)
fproc;
```

## V.7. PORTABILITE - PERFORMANCE.

Le système SAGE a été conçu pour être portable sur n'importe quelle machine.

Pour ce qui est des problèmes d'entrées/sorties liés au système d'exploitation, le système utilise trois sous-programmes très simple écrits en assembleur (de test, lecture et écriture directe sur un port) et dont la reécriture sur un autre système ne poserait aucun problème.

D'autre part seules les caractèristiques de base du terminal (adressage du curseur,inversion vidéo,...) sont à modifier dans le cas d'utilisation d'une autre console. En effet toutes les autres fonctions ont été programmées (cf.§ V.2).

SAGE a été écrit en PASCAL CPM qui est conforme à la norme ISO (mis à part l'utilisation du type "string",l'accès directs sur les fichiers et la gestion dynamique de la mémoire) . D'ailleurs l'emploi d'une option du compilateur peut déceler toute violation de cette norme .

La taille actuelle du système, tous modules confondus, s'élève à 5000 lignes PASCAL.

La version actuelle du système tourne sur un Micral 90-50 sous CPM86.Elle utilise la console standard du Micral en écran "programmeur" et une TAB 132/15 comme console "utilisateur ".

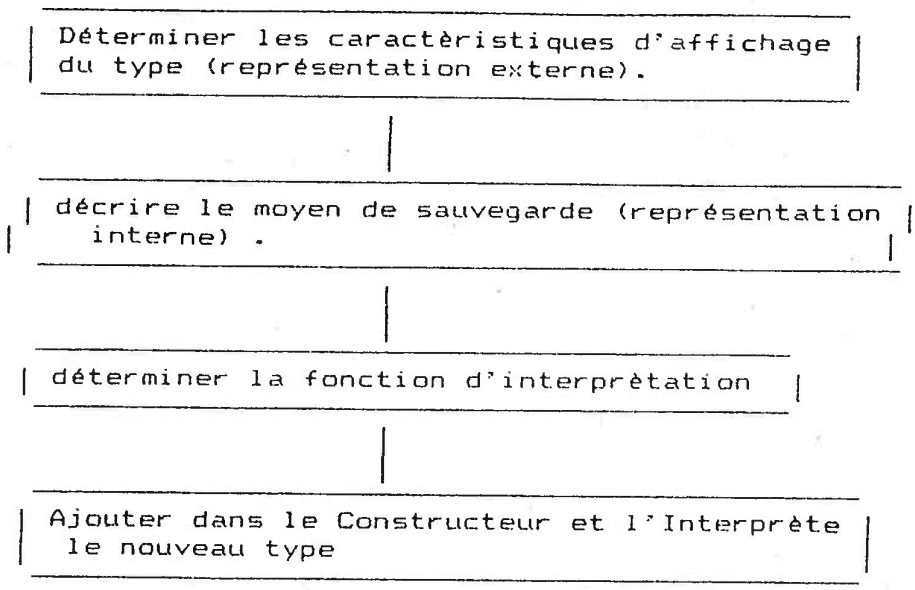
## V.8. EVOLUTIVITE - FLEXIBILITE .

### V.8.1. PRISE EN COMPTE DES EXTENSIONS DE L'UTILISATEUR.

Le système est fléxible à plusieurs égards :

- D'une part au niveau de la description du terminal. Les caractèristiques du terminal sont stockées dans une table qui peut être facilement modifiée à l'initialisation du système.

-D'autre part au niveau de la prise en compte de nouveaux types utilisateurs (qui est une conséquence directe de l'utilisation des types paramètrés dans la spécification cf. chap.IV). L'algorithme d'intégration d'un nouveau type utilisateur peut s'exprimer :




## V.8.2. LE MULTIFENETRAGE.

La section V.7 décrivait les fonctionnalités du module Interprète dans la version actuelle du système,i.e sans prendre en compte la possibilité du "multifenêtrage". Dans ce dernier cas on exige une indépendance totale entre le Cadre et le terminal physique, en particulier la taille du Cadre n'est plus forcément identique à celle de l'écran physique. Dans le cas du "multifenêtrage", l'écran physique se présente comme un ensemble de cadres sur lequel l'utilisateur souhaite faire un certain nombre d'opérations. L'image affichée sur l'écran est telle qu' à tout instant l'écran apparait comme une "mosaïque" de fenêtres, où deux peuvent éventuellemnt se recouvrir partiellement (voir exemple fig.V.3). L'utilisateur aurait alors la possibilité de supprimer ou déplacer une fenêtre, ou agrandir la taille d'une fenêtre ou enfin activer une fenêtre (cf. chap.IV §2.4 et §2.5).

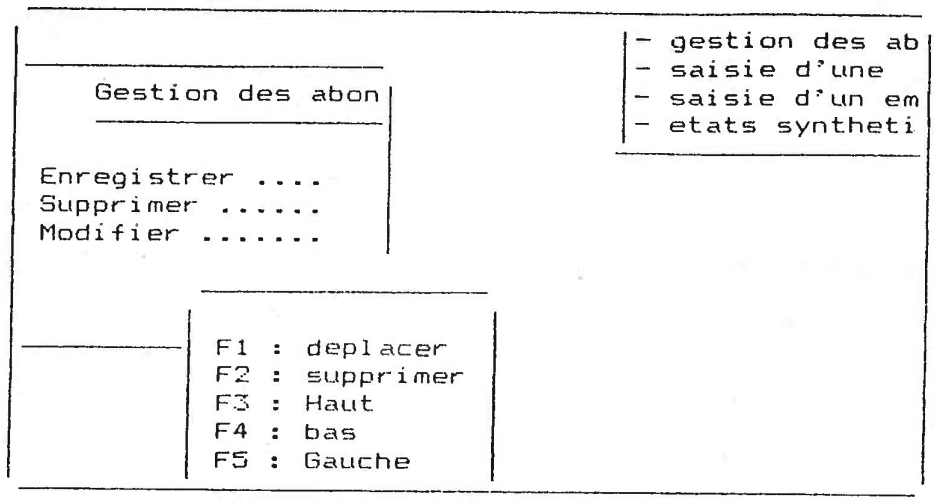



fig. V.3

La prise en compte du multifenêtrage dans le système se ferait simplement en modifiant (compliquant) le module Interprète, les autres modules restent par ailleurs tout-à-fait identiques. En effet outre les procédures "aff-ecr" et "lec-ecr" ; ce module doit gérer d'une part l'ensemble des Cadres-fenêtres en affichage (pour ce qui est des opérations de suppression,déplacement et activation d'un Cadre) et d'autre part l'espace d'affichage lié aux dimensions des fenêtres.

Une description formelle des objets et opérations intervenant dans cette partie est bien avancée (cf. chap.IV § 2.6). Il s'avère que certaines de ces opérations se rapprochent beaucoup de celles du Constructeur ce qui faciliterait certainement la réécriture de ce module.

# CHAP.VI. ETUDE COMPARATIVE DE QUELQUES SYSTEMES DE GESTION D'ECRANS.

## VI.1. INTRODUCTION.

Pour compléter notre étude sur les outils de développement d'applications conversationnelles "pleine-page", nous présentons dans ce chapitre une étude comparative de quelques systèmes (industriels ou prototypes de recherche) d'aide à la programmation d'applications interactives.

Cette étude est organisée de la façon suivante : dans la première section on présente un plan d'étude en indiquant les caractéristiques permettant de comparer les différents systèmes. Dans la deuxième section on discute quelques systèmes existants. Dans la troisième section on présente SAGE face aux critères établis et nous terminons cete section par un tableau récapitulatif. La quatrième section termine cette étude par une série de remarques concluant les résultats de cette étude.

## V.2. PLAN D'ETUDE.

### V.2.1. CARACTERISTIQUES D'UN SYSTEME DE GESTION D'ECRAN.

Pour pouvoir comparer les différents systèmes entre eux on a établi une grille représentant les différents fonctionnalités de tels systèmes. Ces fonctionnalités se répartissent en deux catégories (voir fig.1):

- Celles liées à la "vie" de l'écran :
  . Mode de construction et mise en oeuvre dans une application.

. Facilité d'utilisation: aides à la programmations et services annexes.
. Documentation et archivage.

- D'autres plus générales liées au système :
  . Portabilité.
  . Facilité d'apprentissage.
  . Degré de réalisation.

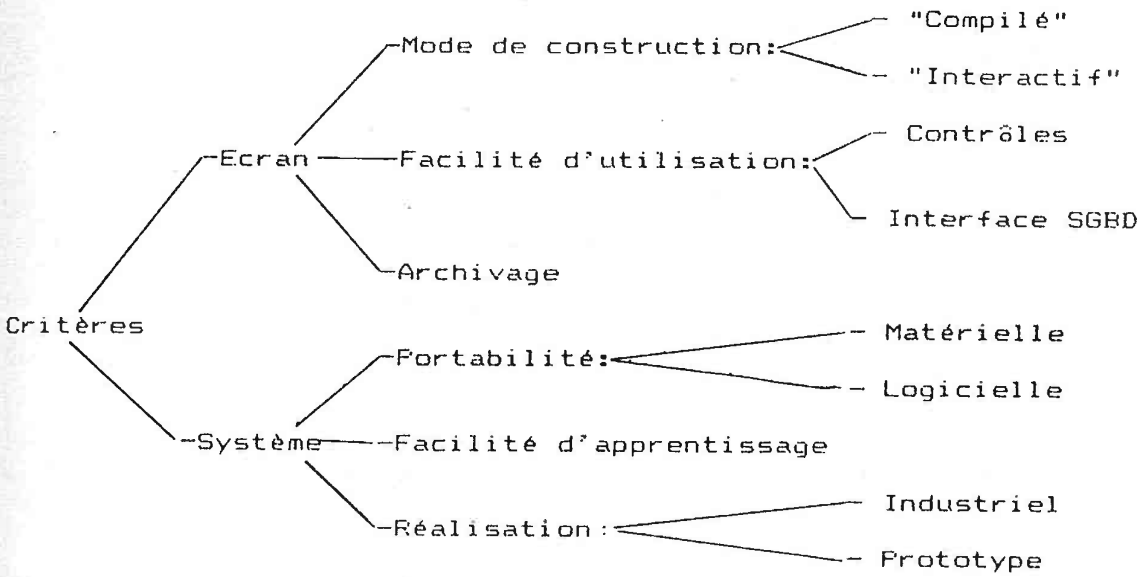



fig. VI.1

Ces caractéristiques sont celles qu'on a pu dégager à partir d'étude des systèmes existants. Avant de présenter les différents systèmes, on discute ci-après la signification de chaque fonctionnalité.

## VI.2.1. ECRAN.

### VI.2.1.1. MODE DE CONSTRUCTION.

En mode "compilé" ou "interactif". Dans le mode "compilé" la description de l'écran se fait dans le programme d'application ; l'exploitation des écrans intervient après compilation ou interpretation) du programme. En général, l'utilisateur définit une (ou des) structures de données dans un langage soit sous forme d'appel de procédures ou à l'aide de macro-instruction. Par contre dans le mode "interactif", le programmeur décrit son écran directement sur la console avec toute la commodité et le

confort qui en résultent sous le contrôle d'un pilote. L'utilisateur dispose d'un éditeur d'écrans plus ou moins sophistiqué lui permettant de créer, supprimer ou déplacer des "fenêtres" dans l'écran.

### VI.2.1.2. FACILITE D'UTILISATION ET SERVICES ANNEXES.

Elle concerne l'aide apportée par le système pour programmer facilement et efficacement les programmes d'applications intéractives (cf. chap.II). Cette aide comprend : le nombre de tâches prises en charge par le système, les facilités de construction de programmes évolutifs et réutilisables..etc.

Les services annexes comprennent traitement des erreurs, interface avec un SGBD et viennent compléter le système d'aide à la programmation. Suivant l'objectif du système ce module est plus ou moins important. Ainsi les systèmes orientés interrogation de bases de données ou mise à jour automatique de fichiers utilisent un dictionnaire de données. Le dictionnaire de données joue le rôle d'interface entre le logiciel de dialogue et les applications utilisateurs.

Le problème des contrôles est un service non moins important. Au niveau de la saisie de données,le contrôle varie du plus simple (test d'une zone élémentaire) au plus complexe (test de cohérence entre zones). C'est cette tâche que le module de contrôle essaie d'automatiser.

### VI.2.1.3. ARCHIVAGE ET DOCUMENTATION.

C'est l'opération suit la description de l'écran. Après avoir décrit son écran, l'utilisateur aura la possibilité de l'archiver, de le rappeler pour ensuite le modifier. Ainsi il pourra utiliser des descriptions existantes. Ce module peut devenir complexe et constituer une véritable base de données principalement au niveau des cohérences.

## VI.2.2. SYSTEME.

### VI.2.2.1. PORTABILITE.

Il s'agit d'une double portabilité.: matérielle et logicielle. La portabilité matérielle se mesure par la possibilité du logiciel d'être utilisé sur plusieurs systèmes différents par :

a) le système d'exploitation (principalement le module d'entrées -sorties).

b) le type de terminal ("intelligence", mémoire, touches de fonctions ..).

La portabilité logicielle est la possibilité du logiciel d'être utilisée dans plusieurs langages. Cette portabilité peut être réalisée directement à l'aide du système d'expolitation. Si non,elle peut faire l'objet d'une importante réflexion au niveau de la spécification et la conception du système.

### VI.2.2.2. FACILITE D'APPRENTISSAGE.

Cette caractéristique est évidement très liée à la caractéristique 2.1.1 (Mode de construction). Ainsi le mode "interactif" est plus facile à utiliser que le mode "compilé".
En mode "compilé", dans le cas de l'extension d'un langage pour la définition d'écrans, le système est plus simple d'apprentissage si l'extension apportée est conforme à la syntaxe du langage.
En mode "interactif", la facilité d'apprentissage est fonction principalement de la performance de l'éditeur d'écrans et du mode de dialogue (commandes, menus, touches de fonctions ..).

### VI.2.2.3. REALISATION.

Il s'agit du degré de réalisation du produit : industriel ou prototype.

Pour l'étude des différents système présentés ci-dessous,on a gardé seulement quatre critères significatifs : mode de construction, aide à la programmation, portabilté, archivage et documentation.

Le tableau récapitualtif présenté en fin de section prend en compte toutes les caractéristiques.

## IV.3. ETUDE DE QUELQUES SYSTEMES DE GESTION D'ECRANS.

La liste présentée dans cette étude est loin d'être exhaustive. En particulier, les multiples systèmes réalisés sur micro-ordinateurs ne sont que brièvement discutés à la fin de la section. Neanmoins cette liste donne une idée globale sur les caractéristiques et concepts utilisés pour développer de tels logiciels.

### IV.3.1 SPF (System Productivity Facility).

SPF (System Productivity Facility) est un produit programme IBM [JOS,81] conçu pour simplifier le développement d'applications

interactives. Il est utilisable sous TSO (Time Sharing Option) le système conversationnel de base.

SPF facilite l'utilsation du système grâce aux trois caractéristiques suivantes :

- dialogue pleine-page.
- possibilité pour l'utilsateur de garder des informations d'une session à une autre.
- un mode particulier de traitement des erreurs.

La fig.2 représente un écran type de SPF (assemblage d'un module).

```
                FOREGROUND ASSEMBLY
ENTER/VERIFY PARAMETERS BELLOW/

PROJECt   ---> SPFDEMO
LIBRARY   ---> MYLIB          ---> TEST  ---> MASTER  --->
TYPE      ---> ASM
MEMBER    ---> TOP
LIST ID   ---> LISTASM                    PASSWORD --->
ASSEMBLER OPTIONS:
      --->LIST,TEST,TERM,RENT
```

fig. VI.2

L'extension majeure du nouveau SPF est l'ensemble des fonctions constituant le gestionnaire de dialogue (Dialog Manager [IBM,81]). Le gestionnaire de dialogue offre à l'utilisateur la possibilté d'utiliser les outils employés de façon interne par SPF pour l'écriture de ses propres applications. SPF devient ainsi une application particulière utilisant le gestionnaire de dialogue.

i. Mode de construction.

En mode "compilé". Le langage de déscription d'écrans est le langage de commandes de TSO qui a été étendu de façon à prendre en compte la définition de "pannels" (écrans). L'utilisateur d'un tel système doit connaître parfaitement le langage de commandes de TSO.

La définition d'un écran se compose de quatres sections:

- attributs : qui définit des caractères spéciaux utilisés pour représenter différents types de champs.
- corps: définit le format de l'écran tel qu'il est vu par

l'utilisateur final et les noms des variables de dialogue.
- initialisations : initialisation des zones.
- actions : définit les actions à exécuter après saisie de l'écran: contrôles,transfert ..etc.

Un exemple de définition d'un écran est donné ci-dessous:

```
/* Il n'y a pas de section attributs. Les attributs par défauts
sont:
% début d'un champ protégé (text) double brillance.
+ début d'un champ protégé (text) brillance normale.
- début d'un champ non protégé brillance normale. */
)BODY
  /* définition du masque d'écran      */

%------------------------EMPLOYEE RECORD ------------------------------

+TYPE OF CHANGE% ---->_TYPECHG + (NEW,UPDATE,OR DELETE)

+EMPLOYEE NAME:
+LAST     % ---->_LNAME           + /* variables de dialogue */
+FIRST    % ---->_FNAME           +

+HOME ADRESS:
+ LINE1 % ---->_ADDR1                                    +
+ LINE2 % ---->_ADDR2                                    +

+HOME PHONE:
+ AREA CODE     % ---->_PHA+
+ PHONE NUMBER % ---->_PHNUM+

     /*    section d'initialisation      */

)INIT
.HELP=PERS032                  /* écran d'aide */
.CURSOR=TYPECHG               /* position initiale du curseur */
If  (&TYPECHG=NEW)
    LNAME=''
    FNAME=''
    ADDR1=''                 /* valeurs initiales des zones */
    ADDR2=''
    PHA=''

    :* section d'activation */

)PROC
 VER(&TYPECHG,LIST,NEW,UPDATE,DELETE,MSG=EMPX210)
 VER(&LNAME,ALPHA)
 VER(&FNAME,ALPHA)
 VER(&PHA,NUM)
 VER(&PHNUM,NUM)
 VER(&PHNUM,PICT,'NNN-NNNN')
)END
```

ii.Aides à la programmation et sevices annexes.

Les valeurs par défaut sont les seules aides à la programmation. Pour les contrôles, l'utilisateur dispose d'une macro-instruction "ver" qui verifie la valeur d'une zone soit relativement à un format soit pour faire des contrôles dynamiques et émission d'un message en cas d'erreur (cf.exemple fig.3 sect. )proc). Il n'existe pas d'interface avec un SGBD. Néanmoins le gestionnaire de dialogue possède une sevice "Tables" qui permet la création et la mise à jour de tables qui sont des fichiers particuliers accessibles simultanément par plusieurs utilisateurs). Une table est décrite comme un "pannel" et peut être adressée à l'aide d'une clé ce qui permet de faire des mises à jour automatiques de fichiers.

iii. Portabilité.

Le gestionnaire de dialogue utilise abondamment TSO sous MVS et n'est utilisable que sur les terminaux IBM 3270 et compatibles. Au niveau des langages aucun exemple n'a été donné quand à son utilisation dans un langage autre que le langage de commande de TSO.

## IV.3.2 FORMS ET SIMILAIRES.

### VI.3.2.1. FORMS-DPS.

Le DPS (Display Processing System) est un logiciel développé par la C.I.I-HB [CII,**] pour permettre à l'utilisateur de construire,maintenir et utilser des "forms" (écrans). Le DPS comprend deux services :

a. Le gestion d'écrans ("Form Processor") qui permet de créer, modifier et maintenir des écrans (en interactif);

b. Le "programmatic Interface" qui réalise l'interface entre l'application et le terminal et entre le terminal et l'opérateur.

i. Mode de construction.

En "interactif". Le gestionnaire d'écrans permet de faire les opérations suivantes:

-créer un écran
-modifier, supprimer ou imprimer un écran déjà creé
-visualiser un écran tel qu'il apparaît au moment de la saisie de données.

-exécuter les contrôles syntaxiques.
-tester un écran.
Toutes ces opérations s'exécutent en conversationnel.

Pour construire un écran, l'utilisateur définit sur la console des champs. Il existe trois types de champs :
- champ fixe ou "text"
- champ variable
- champ invisible.

En plus de ces types, le système permet à l'utilisateur de spécifier d'autres caractéristiques du champ :
-ses attributs: longueur et type du champ, introduction de caractères préétablis.
-un contrôle d'édition sophistiqué en indiquant par exemple qu'un champ doit être ou non complètement rempli, contient une valeur initiale et cadré convenablement s'il est partiellement rempli.

La fig.VI.4 représente un écran type de construction des différents champs:

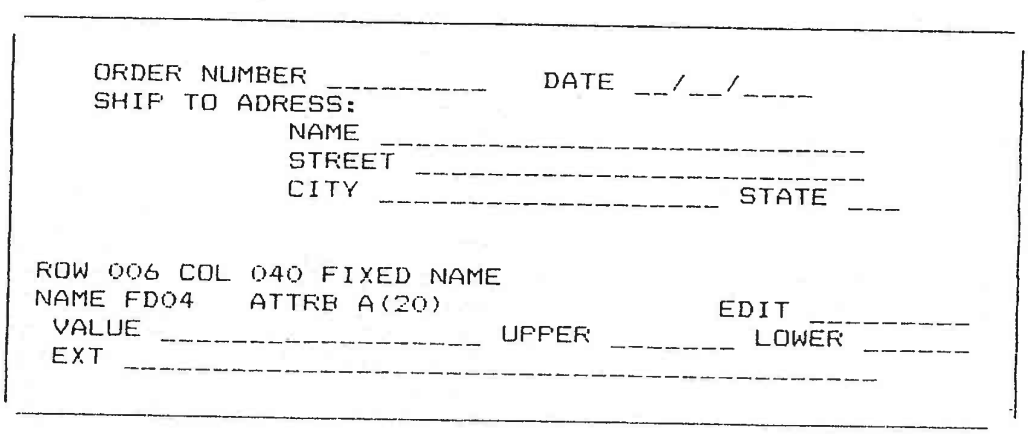



fig. VI.4

ii. Aide à la programmation et services annexes.

L'interface avec le programme d'application est réalisé à l'aide du service "Programmatic Interface". Le P.I consiste en une collection de macro-procédures utilsables par le programme. Les zones de communications doivent être décrites dans le programme utilisateur.

En ce qui concerne les contrôles, ils sont limités aux formats et valeurs limites. Les contrôles inter-champs sont à la charge du programmeur.

iiii. Archivage et documentation.

Le système archive les écrans utilisateurs dans un fichier à accès direct. L'utilisateur peut ainsi appeler un écran pour le modifier (à l'aide des diverses directives du DPS) et ensuite l'archiver. C'est le seul mode d'archivage offert par le système.

iiii. Portabilité.

Les terminaux supportant le système FORMS sont uniquement les terminaux C.I.I :

VIP 7200
DKU 70001
DKU 70002

Au niveau des langages, le système n'est utilisable que pour les langages COBOL, ASSEMBLEUR et PASCAL. D'autre part le système peut être interfacé avec le SGBD SOCRATE.

## VI.3.2.2. AUTRES PROTOTYPES SIMILAIRES.

En général tous les logiciels interactifs de gestion d'écrans se ressemblent dans leur mode d'emploi.On signale ici deux systèmes SIM de la SEMS implanté sur SOLAR-16 [SIM,**] et VIEW-3000 de Hewlett-Packard [HPV,**] en montrant leurs caractéristiques par rapport à FORMS.

SIM se distingue de FORMS d'une part par la façon de représenter les attributs de données sous forme de format. Un format est décrit par une chaine de caractères codés ce qui permet de représenter tous les cas possibles intéressants.
D'autre part l'utilisateur peut sortir une trace récapitulant les champs et les formats utilisés dans un écran.

Dans VIEW en plus du format,l'utilisateur a la possibilité de décrire des contrôles inter-zones lesquelles seront exécutés à la saisie de l'écran. Le système aide l'utilisateur à décrire l'enchaînement des écrans. Enfin l'utilisateur dispose d'une trace analogue à celle produite par SIM et complétée par les informations citées ci-dessus.

## VI.3.3. PASCAL INTERACTIF.

C'est une expérience qui a été réalisée par D.GRIES et J.M.LAFUENTE du Centre de Recherche D'IBM de New-York [LAF,77] et [LAF,78]. Les auteurs proposent une extension de PASCAL pour la définition et la manipulation d'écrans. Les extensions apportées sont essentiellement des règles qui contrôlent le dialogue interactif.

i.Mode de construction.

En "compilé". Le langage de description est le langage PASCAL qui a été étendu de façon à prendre en compte la définition de champs ("items") et écrans ("frames"). Un champ est une variable PASCAL avec d'autres caractéristiques telles que : position du champ sur l'écran, libellé accompagnateur ("text"),format d'entrée (analogue au format FORTRAN), son type (fixe ou modifiable) ..etc. L'utilisateur peut aussi définir des sous-écrans ("subframe") qui sont des regroupement logiques de champs et qui peuvent être utilisés dans plusieurs écrans. Un sous-écran est identique à un Record de PASCAL.

Un écran ("frame") est identique à une procédure sans paramètres. La déclaration d'un écran peut inclure des variables locales, champs, sous-écrans, fonctions et procédures. L'utilisateur peut aussi y inclure des procédures génériques qui sont automatiquement appelées à l'activation de l'écran:

- procédure "initialise".
- la clause "contains" en vue de l'utilisation de sous-écrans.
- un ensemble de règles de dialogue qui décrivent l'interaction programme-utilisateur final, la restriction du dialogue et les conditions de terminaison du dialogue.
- procédure "termine".

Un exemple de définition et l'écran correspondant est présenté ci-dessous:

```
                BANK OF NEW YORK

                  NEW ACCOUNT

Enter information. Hit ENTER when done.
NAME :
PERMANENT ADRESS :
MAILING ADRESS - Same as above ?        * YES
     ENTER ADRESS :                     * NO

STATUS :    * MARRIED
            * SINGLE
```

fig. VI.5

```
Frame F;
var
  ADDR1:key-in char at (0,19) text "PERMANENT ADRESS"
  D3:display text 'MAILING ADRESS'
  M1:menu set of (YES,NO) at (0,16) text 'Same as obove ?'
  ADDR2:key-in char at (0,15) text 'ENTER ADRESS'
contains
  TITLE at (0,17);HEADING;NAME at (5,3);
  ADDR1 at (6,3);D3 at (7,3);M1 at (7,21);
  ADDR2 at (8,6);STATUS at (10,3);
Rules
  require NAME;
  require at least 1 of ADDR1,ADDR2;
  require ADDR1 if YES in M1;
  require ADDR2 if NO in M1;
  with M1: allow only 1 options;
  exclude ADDR2 if YES in M1;
  with STAUS : allow only 1 options;
  terminate if ENT_KEY
end;
procedure initialse;
begin
  reset_frame;M1:=(YES)
end;
endframe;
```

## ii. Aide à la programmation et services annexes.

C'est le rôle des règles de dialogue. En fait ces règles permettent de séparer la description de l'écran de son apparition physique, puisqu'un écran peut changer d'apparence en cours de dialogue avec l'utilisateur final. Les règles de dialogue sont décrites sous forme déclarative. Par conséquent, le programmeur décrit ses règles sans s'occuper de l'ordre de leur traitement.

Ces règles se répartissent en trois catégories:

- Règles de construction : elles spécifient les conditions sous lesquelles les variables ou les "statuts" peuvent être changés par le système.
- Règles de réquisitionnement : qui spécifient les conditions qui doivent être vraies pour terminer le dialogue.
- Règles de terminaison : qui indiquent quand la terminaison est effective.

iii. Archivage et documentation.

Aucune possibilité d'archivage n'a été prévue. La réalisation d'unités (écrans de dialogue) compilés séparement augmenterait de façon sensible la qualité de ce produit.

iiii. Portabilité.

Elle n'a pas été mentionné par les auteurs vu que ce produit n'a pas été complètement implémenté.

VI.3.3. SCREEN-RIGEL.

SCREEN-RIGEL est un sous-système de RIGEL (un langage de haut niveau de définition et de manipulation de bases de données relationnelles). De ce fait il dispose de tous les outils du langage RIGEL pour accèder aux bases de données, décrire les contrôles et l'enchaînement des écrans comme dans une application interactive classique.

i. Mode de construction.

En "compilé". Le langage de description est le langage RIGEL. La définition d'un écran ("frame") se compose de trois sections:

-section de données : Dans cette section sont déclarées toutes les zones intervenant dans l'affichage.

-section format : qui décrit comment apparaissent les différents champs dans l'écran.

-section d'activation : Dans cette section sont décrites les les actions à exécuter après la saisie de données.

Pour décrire les types de données,le système utilise tous les types de RIGEL (y compris le type relation). La définition du format est faite à l'aide de procédures de "formatage" avec paramètres. Les actions de la troisième section servent principalement à décrire des contrôles globaux et se terminent par l'exécution d'une des actions suivantes : Continue, Sortie, ou Rejet. Un exemple de définition d'un écran est donné ci-dessous :

```
          Edit Personal Information

Name : ............          Occupation : ..........
Age  : ...                   Marital staus :
                                          Single
Salary : ......                           Married
Pay status :                              Other
            Exempt
            Nonexempt
            Contract
```

fig. VI.6

```
edit-person:frame
     PERSON'record-type;
format
  center('Edit personal Information')
  column(2,
    output(name);
    modify(age,format:'DDD',check:ageCheck);
    modify(salary,format:'5D.2D',check:salaryCheck);
    modify(pay Status,label:'pay status');
    modify(occupation);
    modify(maritalStatus,label:'marital status');
    output(ssn%,'format:'3D-2D-4D')
  )
end;
```

ii. Aide à la programmation et services annexes.

C'est le meilleur aspect développé par SCREEN-RIGEL. L'aide apportée par le système se concrétise en quatre points:

1. Dans la partie format, le programmeur n'est pas obligé de donner tous les paramètres requis par la fonction.

2. Du fait que c'est un sous-système de RIGEL, SCREEN-RIGEL peut accèder directement aux bases de données pour en faire la mise à jour ou l'interrogation.

3. Pour décrire un écran dont la structure est analogue à une

relation existante, le programmeur dispose d'une clause "même type que" (cf. exemple fig.6). Cette clause a un double avantage: a) elle simplifie la description de l'écran. b) elle sert à maintenir les cohérences : en effet si une modification est apportée à une relation, elle est automatiquement translatée à l'écran.

4. Ce dernier point concerne les contrôles et résulte directement des points 1. et 2. ci-dessus. En effet le programmeur peut inclure dans le format d'un champ le nom d'une procédure qui sera exécutée à la saisie de ce champ. Cette procédure aura été définie dans le dictionnaire de données.

Pour d'autres contrôles,SCREEN-RIGEL utilise les instructions conditionnelles de RIGEL : if,while, switch, ...etc.

iii. Archivage et documentation.

SCREEN-RIGEL utilise le Dictionnaire de Données de RIGEL. Le Dictionnaire de Données contient des informations décrivant des objets de base (relations,..), des modules (collection de types, variables et procédures) et les écrans. Il est alors possible par un mécanisme de formattage par défaut d'atteindre une indépendance entre applications et données, i.e certains changements au niveau de l'écran n'affectent pas l'application.

iiii. Portabilité.

D'après les auteurs, le système peut être porté sur n'importe quelle machine. Le système sait "s'adapter" aux configurations de chaque terminal qui sont d'ailleurs des paramètres du système. les auteurs n'ont mentionné aucune expérience effective. La portabilite au niveau des langages se pose au niveau du langage RIGEL, problème bien connu des concepteurs de bases de données.

## VI.3.4 GESCRAN-CONSCARN.

GESCRAN est un progiciel qui a été développé par l'équipe de B.MEYER [GES,80] au Centre de Recherches d'EDF.

i.Mode de construction.

GESCRAN a été initialement prévu pour être utilisé en mode "compilé". GESCRAN permet à l'utilisateur, par une série d'appels à des sous-programmes FORTRAN, de définir un ensemble d'objets abstraits appelés "écrans", de créer dans ces écrans des fenêtres rectangulaires, d'initialiser le contenu de ces fenêtres, de spécifier et modifier leurs caractéristiques (protection, brillance, couleur..),d afficher les écrans sur le terminal et de prendre en compte les différents types de données entrés par

l'utilisateur au terminal.

Un autre progiciel, CONSCRAN, a été développé pour utiliser de façon efficace GESCARN. En effet, l'utilisation directe de GESCRAN (mode "batch") s'est révélée très lourde. CONSCRAN permet de faire les mêmes opérations que GESCRAN et en mode interactif. En fin de construction, CONSCRAN génére un sous-programme FORTRAN constitué de l'ensemble des appels à GESCRAN pour la construction de cet écran.

Un exemple de construction d'un écran et le sous-programme correspondant est donné ci-dessous:

```
Enregistrement Informations
**************************

        NOM      ==> ...........
        PRENOM   ==> ............
```

```
      SUBROUTINE CONST
      INTEGER ECRAN
      INTEGER TITRE,QUES
      INTEGER REPN,REPPN
C  définition de l'écran
      CALL DEFGE(ECRAN)
C  Création de la fenêtre TITRE
      CALL CREFGE(TITRE,ECRAN,1,2,2,20)
      CALL PROFGE(TITRE)
      CALL BRIFGE(TITRE,'B')
      CALL UNISOR(TITRE)
      CALL ECRTIT('/Enregistrement Informations/')
      CALL ECRTIT('/**************************/')
C  création de la fenêtre QUES
      CALL CREFGE(QUES,ECRAN,10,14,20,45)
      CALL PROFGE(QUES)
      CALL BRIFGE(QUES,'B')
      CALL UNISOR(QUES)
      CALL ECTEX('/ NOM      ==>/')
      CALL RETLIG
      CALL ECRTEX('/PRENOM   ==>/')
C  création de la fenêtre REPN
      CALL CREFGE(REPN,ECRAN,10,10,46,60)
C  création de la fenêtre REPPN
      CALL CREFGE(REPPN,ECRAN,13,13,45,60)
      RETURN
      END
```

fig. VI.7

ii. Aide à la programmation et services annexes.

Un système d'aide à la programmation fondé sur les notions de programmation orientée objet et visant à structurer les programmes de dialogue est en cours de réalisation [MEY,82].

iii. Archivage et documentation.

Il est limité aux écrans. Auncun interface avec un SGBD n'a été réalisé.

iiii. Portabilité.

GESCRAN n'est actuellemnt utilisable que sur les terminaux IBM 3270 et compatibles. La portabilité logicielle est réalisée directement par le système d'exploitation. En effet, les sous-programmes de GESCRAN peuvent être appelés par tous les langages disponibles sur le système.

La nouvelle version de GESCRAN [GES,83] apporte des améliorations nettes par rapport à l'ancienne. En effet, il y a indépendance totale entre le terminal physique et l'écran grâce au concept de multi-zones (dans la terminologie de GESCRAN). D'autre part les zones ne s'effacent plus et peuvent éventuellemnt se recouvrir.

## VI.4. LE SYSTEME SAGE.

Rappelons que l'objectif du système SAGE est de fournir des outils et une méthode de spécification et de construction des applications interactives en partant des principes suivants :

- séparer le dialogue des traitements.
- faciliter la description d'écran à l'aide d'un éditeur interactif.
- permettre le stockage et la réutilisation de descriptions.
- faciliter la mise en oeuvre des écrans dans un programme d'application.

i. Mode de construction.

En interactif. Le "langage de description" repose sur les notions d'Article, Sous-Cadre (et les concepts déduits du Sous-Cadre: Tableau et Menu) et Cadre logique (cf. § III.3).
Ces notions ont été choisies ainsi pour être le plus possible proche de la conception de l'utilisateur.

Pour décrire son Cadre, l'utilisateur dispose d'un éditeur qui lui permet de construire progressivement la structure finale

du Cadre. Les deux caractèristiques principales de cet éditeur sont : l'utilisation du mécanisme de désignation directe (cf. § III. 4) et la possiblité d'une documentation "en-ligne".

ii. Aides à la programmation.

La conception du système SAGE repose sur une méthode de programmation des applications conversationnelles, celle qui a été exposée au § II.2.3 . L'aide apportée par le système pour écrire les programmes d'applications se concrètise sur trois points :

- simplification de l'interface avec le programme utilisateur (limitée à deux procédures, i.e affiche et saisie).

- garder la logique du programme notamment par la génération de l'équivalent de la description du Cadre dans le langage de programmation de l'utilisation et d'une procédure d'interprétation de cette description ( § III.5).

- Automatisation des procédures de Contrôles et d'éditions.

iii. Archivage et documentation.

Cette caractéristique a reçu une attention toute particulière dans le développement de SAGE. En effet à chaque niveau de description, l'utilisateur dispose d'un catalogue des éléments déja définis qu'il peut consulter ou mettre à jour . Cette façon de procéder évite, d'une part à l'utilisateur de redéfinir des entités existantes et permet, d'autre part, d'éliminer les erreurs de manipulation lorsque plusieurs applications utilisent des entités communes.

La documentation est présente dans le système surtout au niveau de l'édition (cf. commandes SOS,TRAce papier ...).

iiii. Portabilité.

Le système SAGE a été conçu pour être porté sur n'importe quelle machine (cf. § IV.7). Cette caractèristique a été, en effet, étudiée et maintenue depuis la spécification jusqu'à la réalisation du système .

VI.5. TABLEAU RECAPITULATIF.

| | ECRAN | | | | | SYSTEME | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Mode de Construction | | Aide à la programm. | | Archivage | Portabilité | | Facilité d'apprentissage | Finalité | |
| | "Batch" | "Interactif" | Contrôles | SGBD | | matérielle | logicielle | | Prototype | Industriel |
| GPF | Oui | | Limités | non | Oui | non | difficile | Lourd d'emploi. destiné à des spécialiste du L.C TSO | | oui |
| FORMS | | oui | Limités | non | oui | non | oui | Oui pour une première utilisation | | oui |
| VIEW.3000 | | oui | oui | oui ? | oui | non | oui | " | | oui |
| Pascal Interactif | oui | | oui | non | non | ? | utilisable uniquement en Pascal | Extensions conformes à la syntaxe Pascal | oui | |
| Screen Rigel | oui | | oui | oui | oui | oui ? | non | exige la connaissance de Rigel | oui | |
| GESCRAN-CONSCRAN | oui | oui | non | non | limité | oui ? | oui | à l'aide de Conscran | oui | oui |
| SAGE | | oui | oui | non | oui | oui | non | l'écran est décrit pas à pas | oui | |

## VI.5. DISCUSSION.

L'étude comparative de ces quelques systèmes nous mène à émettre les remarques suivantes :

1. La plupart des produits présentés dans cette étude (4 sur 6) sont à vocation industrielle. Ceci explique le désintérêt pour des caractéristiques telles que la portabilité (caractéristique qui tient une importance fondamentale dans un système tel que SAGE).

2. La plupart des systèmes sont caractérisés par un mode de construction de type "batch", ce qui parait un peu primitif par rapport au but poursuivi. Ceci est dû principalement à la jeunesse de cette activité du génie logiciel et a une idée généralement admise : on maitrise mieux une technique lorsqu'on peut la construire à partir de programmes séquentiels. Idée qui est actuellement largement dévolue (cf. éditeur structurels, programmation par exemple,..) [STE,83].

3. Tous les systèmes (à part Pascal-interactif et à un degré moindre Screen-Rigel) n'intègre pas de méthodes de description des applications conversationnelles, ni de modèle préalable de prise en compte de l'utilisateur final.

4. Tous les systèmes (y compris SAGE) se sont limités à la description du dialogue dans des domaines précis : informatique de gestion, EAO, dialogue par menus successifs et ignorent en quelque sorte les nouvelles exigences des applications interactives (écran haute résolution, multifenêtrage, graphisme..). D'autre part les travaux récents dans ce domaine se situent encore aux stade de recherche (cf. [SHA,83] et [COU,84]) ce qui montre la complexité certaine de ce problème. Ce retard peut, aussi, s'expliquer par l'aspect souvent considéré comme mystérieux de certains outils interactifs tels que les éditeurs de textes [M&V,82].

5. Les systèmes présentés n'offrent pas souvent un degré elevé d'abstraction. Cette contrainte influe beaucoup sur le dégré de généralité du système. Peut-on utiliser facilement l'un de ces systèmes pour construire, par exemple, un éditeur de texte?

Dans SAGE ce problème a été étudié avec soin grâce notamment à une description hiérarchique précisant à chaque pas les différentes options prises pour définir un objet.

# CONCLUSION GENERALE.

La réalisation du système SAGE s'est faite dans un cadre qui peut être caractérisé par les quatre aspects suivants :

a. Intégration de méthodes :

Les outils proposés doivent faciliter la programmation des applications conversationnelles en suivant une méthode. A chaque étape du développement nous devons nous assurer que les décisions prises n'altèrent pas ce schéma.

b. Facilité d'utilisation :

Cet aspect concerne la prise en compte de l'utilisateur final dans le développement des supports et scénarios de dialogue. La solution prise dans le système SAGE est celle qui consiste à donner à l'utilisateur la possibilité de définir lui même et la structure et la forme du dialogue.

c. Facilité d'apprentissage et réutilisation :

Cette caractéristique est liée au premier aspect (méthodes). En effet, le mode de construction des différentes entités d'affichage suit un processus descendant qui nécessite à chaque étape l'archivage des entités définies.

d. Portabilité :

Cet aspect concerne la portabilité du système par rapport au matériel et système d'exploitation. L'option prise dans SAGE est d'écrire des utilitaires qui réalisent les fonctions usuelles d'un terminal moderne.

Dans le système SAGE, nous ne prétendons pas satisfaire toutes ces caractéristiques, ce qui est d'ailleurs impossible. En effet, si ces caractéristiques sont pour la majorité complémentaires, certaines sont par contre peu compatibles entre elles. Ainsi, en donnant la possibilté à l'utilisateur de définir lui même les supports du dialogue, on impose des contraintes au programme d'application. De la même façon, en "obligeant" l'utilisateur de réutiliser des entités existantes (ce qui augmente certainement la productivité), on limite en partie la souplesse d'utilisation.

Il nous fallait donc trouver un compromis entre ces différentes caractéristiques grâce, notamment, à une structure hiérarchique à plusieurs niveaux : Pilote, Constructeur,

Bibliotècaire, Interprète, Utilitaires et les interfaces entre ces différents modules. Notons aussi les importants services rendus par l'utilisation des types abstraits pour identifier les différents composants du système et leur comportement global.

Les idées et outils utilisés dans la réalisation du système SAGE sont pour une partie assez classiques et à ce point de vue critiquables :

-Utilisation du langage PASCAL. PASCAL s'est, en effet, révélé inadapté pour manipuler la structure intrinsèque (de type arborescence) des objets d'édition et d'affichage.

-Utilisation d'un matériel (clavier-écran) à fonctions limitées ne permettant pas d'envisager des perspectives de dialogue plus sophistiqués (graphisme, "bitmap", souris,..).

C'est un des points que nous comptons réaliser à court terme.

Par contre, pour une autre partie, les idées utilisées dans SAGE nou parraîssent très importantes dans le développement d'interfaces homme-machine et ne semblent pas être bien maitrisées par d'autres systèmes. Ces idées concernent principalement :

-L'intégration de méthode de programmation des applications et de construction du dialogue.

-Forte interactivité et prise en compte de l'utilisateur final. Cette idée exprime le fait simple suivant (constaté par plusieurs concepteurs du dialogue) : puisqu'il est difficile (voire impossible) de prévoir à l'avance la meilleure forme du dialogue qui satisfera l'utilisateur, une meilleure solution pour y parvenir est de donner la possibilité à l'utilisateur de définir lui même cette forme.

-La réutilisation d'entités d'affichage. Cette idée est une simple application du principe bien connu en génie logiciel, i.e penser à construire un programme c'est aussi penser à le modifier et à le réutiliser.

Parmi les extensions futures du système SAGE, nous comptons utiliser les spécifications formelles, les idées clés du système et l'expérience acquise dans sa réalisation pour développer un outil général d'aide à la construction d'applications interactives. Par outil général on entend :

-Un outil utilisable aussi bien pour l'édition de texte, le graphique, l'éditions d'objets structurés ..etc. Ce qui nécessite un degré élevé d'abstraction.

-Un outil (nécessairement interactif) qui implique le plus possible l'utilisateur final dans la conception du dialogue. On

pourrait imaginer que le processus du dialogue se répartit entre deux niveaux : le niveau "programmeur" (logique) où l'on prend des décisions générales sur la forme du dialogue et un niveau "utilisateur" qui "instanciera" le dialogue du niveau programmeur pour son besoin particulier. Ce passage pourrait être réalisé à l'aide d'un système "expert" du dialogue qui puiserait ses connaissances dans une base de "styles" de dialogue (c.à.d les différentes de mise en oeuvre d'une structure d'affichage ou saisie). Un exemple type montrant l'interêt de ce double mécanisme est la notion de menu. Un menu est défini au niveau logique (programmeur) comme un un ensemble d'options dont une sert de réponse. Au niveau utilisateur un menu peut être mis en oeuvre de plusieurs façons : 1) en tapant le nom de l'option choisie ; 2) en séléctionnant une option à l'aide d'un moyen quelconque de désignation ou enfin en appuyant sur une touche de fonction (dont on aurait, au préalable, établi la correspondance entre la suite des options et le nombre de touches existantes).

Ce projet peut être comparé dans ses objectifs aux projets (en cours de développement) DESCARTES [SCH,83] et l'interface usager construit pour l'atelier Adèle [COU,84] mais s'en distingue surtout par le degré élevé d'abstraction, la forte interactivité et la prise en compte de l'utilisateur final.

## Réferences bibliographiques.

[ABR,80] : R.ABRIAL, S.A.SHUMAN, B.MEYER : "A specification language ". in On then construction of programs. Ed. C.A.R HOARE. Cambridge Univ. Press 1980.

[BOC,84] : P.BOCCARD : "Un Système d'Aide à La gestion d'Ecran". Thèse CNAM. Juin 1984.

[BRN,82] : J.W.BROWN : "Controlling the complexity of menu network" . Comm. of ACM Jui. 1982.

[BYT,80] : BYTE Magazine : Numéro special sur Small-Talk. Août 1981.

[CHT,82] : A.CHTAYTI : "Vers un système d'aide à la programmation et la gestion des applications conversationnelles pleine-page" Rapport DEA. DER-EDF CLAMART Sept. 1982.

[COT,71] : COTTON,I.W : "Languages for graphic attention-handling". In advanced Computer Graphics, Plennus Press 1971.

[COU,84] : J.R.COUTAZ : "Adèle et la médiateur compositeur". Actes 2ème congrès génie logiciel. Nice Juin 1984.

[DAT,78] : DATA GENERAL : Serie NOVA, "SCRENN-HANDLING" 1978.

[DIJ,76] : E.W.DIJKSTRA , A discipline of programming. Prentice Hall 1976.

[DIG,80] : DIGITAL RESEARCH : "Display Manager" 1980.

[DUF,80] : J.F.DUFOURD : "Types abstraits, modèles relationnels et langage SIMULA". Colloque "Les bases de donées : modèles, mise en oeuvre et évaluation". Chapitre Français de l'ACM Univ. Paris VI. 14-15 Juin 1979.

[FAU,83] : B.FAULLE-G.THOMAS : "Analyse des systèmes conversationnels". 01 Informatique No.137 pp.40-46.

[FIN,79] : J.P.FINANCE -J.C.DERNIAME. "Types abstraits : spécification, utilisation et réalisation". Ecole d'été MONASTIR 1979.

[F&S,83] : J.P.FINANCE-A.SCHAFF: "Aspects ergonomiques des applications conversationnelles". Rapport interne CRIN 1983.

[FRI,84] : B.FRIMAN : "MGEN : A generator for Menu driven programs". Actes congrès Soft. Eng. ORLANDO Mars 1984.

[GES,80] : E.Audin/G.Brisson/B.Meyer "Gestion d'écrans alphanumériques"

Note AL.22 .Version 3. EDF-DER CLAMART.

[GES,83] : B.MEYER/G.BRISSON/F.VAPNE "Gestion d'écrans alphanumériques" Note AL 22. Version 4. EDF-DER CLAMART.

[GOG,79] : J.A. GOGUEN - J.W. TATCHER - E-W. WAGNER " A initial algebra to specification, correctness and implementation of abstract data types". Acta informatica 1978.

[GOL,83] : A.GOLBERT : "SMALL-TALK the language and it's implementation". Adisson-Wesley Publishing 1983.

[GUT,80] : J.GUTTAG - J.J HORNING "The algebraic specification of abstract data types". Acta informatica, vol. 10, 1978.

[GUT,80] : J.GUTTAG - J.J.HORNING "Formal specification as a design tool" 7eme PEOPL ACM 1980.

[HER,82] : N.HERTSCHUH-A.SCHAFF : "Etude préalable d'une application transactionelle". Rapport CRIN 1982.

[HPV,82] : HEWLETT-PACKARD : VIEW-3000 "manuel d'utilisation" 1982.

[IBM,69] : IBM "Course writer" user's manuel. 1969.

[IBM,81] : IBM Dialog Management Service Sc 34-2036 1981.

[JAC,83] : R.JACOB : "Using formal specification in the design of Human-Computer interface". CACM Avril 83 Vol.26 NO.4.

[JON,76] : C.B.JONES : "Software developpement : a rigourous approch". Prentice Hall 76.

[JOS,81] : PH.JOSLIN : SPF "System Procduvity Facility ". IBM System Journal No 4 1981.

[KAI,82] : P.KAISER-I.SETTINA : "A dialog generator". Software practice Vol. 12 1982 .

[LAF,77] : J.M.LAFUENTE : "The specification of a data directed interactive User-Computer dialog". Cornell University PH.D 1977.

[LAF,78] : J.M.LAFUENTE-D.GRIES : "Language facilities for programming User-Computer dialog". IBM J.RES & Dev. No.2 Mars 1978.

[MAR,73] : J.MARTIN : "Man Machine Interface". Prentice Hall 1973.

[MEV,82] : N.MEYROWITZ, A.VAN DAM : "Interactive editing systems" Part 1. Comp. Surv. Vol. 14, NO. 3, Sept. 82.

[MEY,82] : B.MEYER : "Un environnement conversationnel à deux dimensions " . Actes journées BIGRE 1982.

[MEY,84] : B.MEYER - J.M.NERSON : CEPAGE "Un éditeur structurel pleine-page". Actes congrès CGL2 Nice Juin 84.

[NEW,72] : W.M.NEWMAN : "Display procedures " . CACM Vol.14 No 10 Oct.72.

[PIE,81] : GOLDSTEIN : "Extending Objects oriented programming in Small-Talk " Xerox. Mars 1982.

[ROW,82] : L.A.ROW-K.A.SHONES : "Programmming language constructs for screen definition". IEEE Trans. on Soft. Eng. Vol SE-9 Jan. 83.

[SAN,78] : E.SANDEWALL : "Programming in the interactive environnement : the LISP experience ". ACM comp. surv. 10,1,Mars 78.

[SEM,80] : SEMS-SOLAR 16 Notice SIM : "Manuel de référence ".

[SCH,81] : A.SCHAFF : "Assistance à la production de logiciel transactionnel ". Rapport CRIN 81-R-005.

[SHA,83] : M.SHAW : DESCARTES " A programming language approch to interactive display". ACM 83.

[SHE,80] : B.SHNEDERMAN : Software Psychologie "Human factors in computing information systems" .Winthrop Inc. Cambridge, Massaschussets 1980.

[STE,83] : L.STEEL : "ORBIT : An applicative vue of object-oriented programming" . Actes Col. " Integrated Interactive Computing Systems" . North-Holland Publishing 1983.

[STU,84] : R.STUDER : "Abstract models of dialogue concepts ". Actes congrès Software Enginiering Orlando . Mars 84.

[SUF,82] : B.SUFFRIN : "Formal specification of a display oriented text editor". SCP 1 1982 .

[SUT,73] : J.SUTTON-R.SPRAGUE : "A study of display generator and management interactive business applications". IBM.Res Labo. Nov. 1973.

[WAS,81] : A.I.WASSERMAN : PLAIN "An algorithmic language for interactive information system". IFIP 81, North-Holland.

[WAS,84] : A.I.WASSERMAN . Rapide/Use "Un outil pour construire des S.I interactifs". Actes CGL 2 Nice. Juin 84.

[WIN,79] : WINOGRAD : "Why programming languages are obsolete ". Soft. Practice 1979.

[ZLO,75] : M.ZLOOF : Query-by-exemple. IBM Thomas J.Watson research Center Yorktown. New-York 1975.

institut
national
polytechnique
de lorraine

Le Président,

VRéf. : Scol.

AUTORISATION DE SOUTENANCE DE THESE DE DOCTORAT 3ème CYCLE

-:-:-:-:-:-:-:-:-:-:-:-:-

VU LE RAPPORT ETABLI PAR :

**Messieurs FINANCE J-P. - Professeur -**

**SCHAFF A. - Maître-Assitant -**

le Président de l'Institut National Polytechnique de Lorraine autorise :

**Monsieur CHTAYTI Ahmed**

à soutenir, devant l'I.N.P.L., une thèse de Doctorat intitulée :

**"Etude d'un système d'Aide à la Programmation d'Applications conversationnelles. Spécification et réalisation d'une maquette."**

en vue de l'obtention du titre de DOCTEUR 3ème CYCLE

Spécialité **"INFORMATIQUE"**

Fait à NANCY, le 23 Octobre 1984

Le Président de l'I.N.P.L.

M. LUCIUS