MASTER EN SCIENCES INFORMATIQUES

Etude et réalisation d'un générateur d'écrans en langage C

Geltmeyer, Yves

Award date:
1987

Awarding institution:
Universite de Namur

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

Etude et réalisation d'un générateur d'écrans en langage C.

Mémoire présenté par
Yves GELTMEYER
en vue de l'obtention du
diplome de licencié
en informatique.

Année académique 1986 – 1987.

Je remercie particulièrement Monsieur GIGOT pour l'aide constante qu'il m'a prodiguée et pour le temps qu'il a consacré pour la réalisation de ce mémoire.

J'exprime également ma gratitude à Monsieur BOULANGER pour l'acceuil agréable qu'il m'a réservé au sein de sa société.

Mes remerciements vont aussi à mon épouse pour sa patience et son soutien moral.

CSGEN : C SCREEN GENERATOR

SOMMAIRE.

TABLE DES FIGURES.

CHAPITRE 1 : INTRODUCTION.

1.1 La société.

Fin août 1986 la société Prologic souhaitait étendre son département Développement de Logiciels, sous les différents systèmes d'exploitation qui accompagnent les systèmes informatiques qu'elle commercialise, c'est-à-dire MS-DOS, CCPM-86 et UNIX.

Dans une telle approche, la portabilité a été pressentie comme fondamentale, c'est la principale raison qui a amené les responsables de la société à choisir le langage C pour développer ses applications.

Prologic développe presque exclusivement des programmes de gestion, impliquant l'utilisation d'utilitaires tels que gestionnaires de fichier, SGBD, gestionnaires d'écrans,...

Sous MS-DOS, le Lattice C est disponible avec son gestionnaire de fichiers, DBC3-ISAM. De même sous UNIX et CCPM-86, divers utilitaires sont disponibles, par exemple C-ISAM. Par contre, le nombre de gestionnaires d'écrans disponibles sur le marché est beaucoup plus limitée, ou bien ils sont peu puissants, ou encore d'un coût prohibitif. Il faut également souligner que beaucoup de gestionnaires d'écrans disponibles sous MS-DOS sont soit associés au langage BASIC, soit non portables à cause d'appels trop nombreux au BIOS des ordinateurs compatibles avec l'IBM-PC. Enfin, sous UNIX, le transfert de programmes sur des machines de modèles ou de constructeurs différents nécessite souvent une recompilation car les environnements hardware (CPU,...) sont très disparates.

1.2 Etude des besoins.

Ces constatations ont amené les responsables de Prologic à envisager la réalisation de leur propre gestionnaire d'écrans. Un cahier de charge a alors été dressé, en voici les points principaux :

Performance

Les utilisateurs d'un Personal Computer sont habitués à voir une gestion d'écran très rapide et cette approche doit être maintenue.

Versatilité

Il ne faut imposer ni restrictions, ni "lourdeur" au programmeur qui utilisera le logiciel.

Simplicité

Les notions de base et l'utilisation du gestionnaire doivent être rapidement assimilables par le personnel.

Ergonomie

Le gestionnaire doit respecter (si possible imposer) une grande convivialité pour l'utilisateur final.

Portabilité

Le gestionnaire ne doit utiliser aucune particularité d'un système d'exploitation et doit pouvoir être utilisé sur une grande variété de terminaux.

1.3 Historique.

Le développement de la première version du générateur d'écrans s'est terminée en décembre 1986. Elle a été baptisée "CSGen", les initiales de "C Screen GENerator".

Bien qu'opérationnelle et employée au sein de la société, cette version a montré quelques inconvénients qui seront énoncés au paragraphe 4.2. En effet, les écrans y sont vus comme une simple suite de caractères affichés sur l'écran ou lus à partir du clavier.
La lecture de l'ouvrage "L'informatique conversationnelle" de B.Faulle nous a éclairé quant au rôle de pilote que possèdent les écrans dans une application conversationnelle.
La fusion des concepts dégagés par B.Faulle avec ceux qui ont été introduits par la première version de CSGen a abouti à la réalisation de la seconde version du générateur d'écrans.

1.4 Aperçu du travail.

Le chapitre 2 commence par énoncer quelques notions élémentaires liées à la gestion d'écrans; au paragraphe 2.9, nous énonçons un exemple qui servira de base au développement des autres concepts liés à la gestion d'écrans.

Le chapitre 3 décrit les concepts et le fonctionnement de CSGen utilitaire appuyé par un exemple de programme utilisant le générateur d'écrans.

Le chapitre 4 reprend la démarche du chapitre 3, mais appliquée à la version superviseur de CSGen.

CHAPITRE 2 : CONCEPTS GENERAUX UTILISES PAR CSGEN.

Les relations homme-machine constituent une problématique importante à l'heure où presque chaque individu utilise un système informatique. Il est impératif que ces relations soient étudiées de manière à éviter à l'utilisateur une trop grande fatigue visuelle ou nerveuse.

Etant donné que l'interface homme-machine se compose d'un clavier et d'un écran, la conception des écrans présentés à l'utilisateur doit permettre une meilleure communication. On peut dés à présent dégager les caractéristiques suivantes :

- Grande lisibilité des écrans.
- Utilisation rationnelle du clavier.
- Permanence d'un "style" de présentation pour toutes les applications utilisées par une personne.

2.1 Conception générale des images écran.

Une image écran devrait comporter au moins les sous-ensembles suivants :

- Un sous-ensemble TITRE.
- Un sous-ensemble CORPS D'ECRAN.
- Un sous-ensemble MESSAGES.
- Un sous-ensemble OPTIONS.

Ceux-ci pourraient se répartir géométriquement de la façon illustrée à la figure 2.1.

Le sous-ensemble TITRE :

> Il contient des informations qui doivent toujours être présentes sur l'écran : la date, l'heure,... et, bien entendu, la description de l'image-écran affichée.

Le sous-ensemble CORPS D'ECRAN :

> Il contient les informations affichées par le programme et les réponses entrées par l'opérateur. Le corps d'écran est le seul sous-ensemble où se déroule un réel "dialogue".

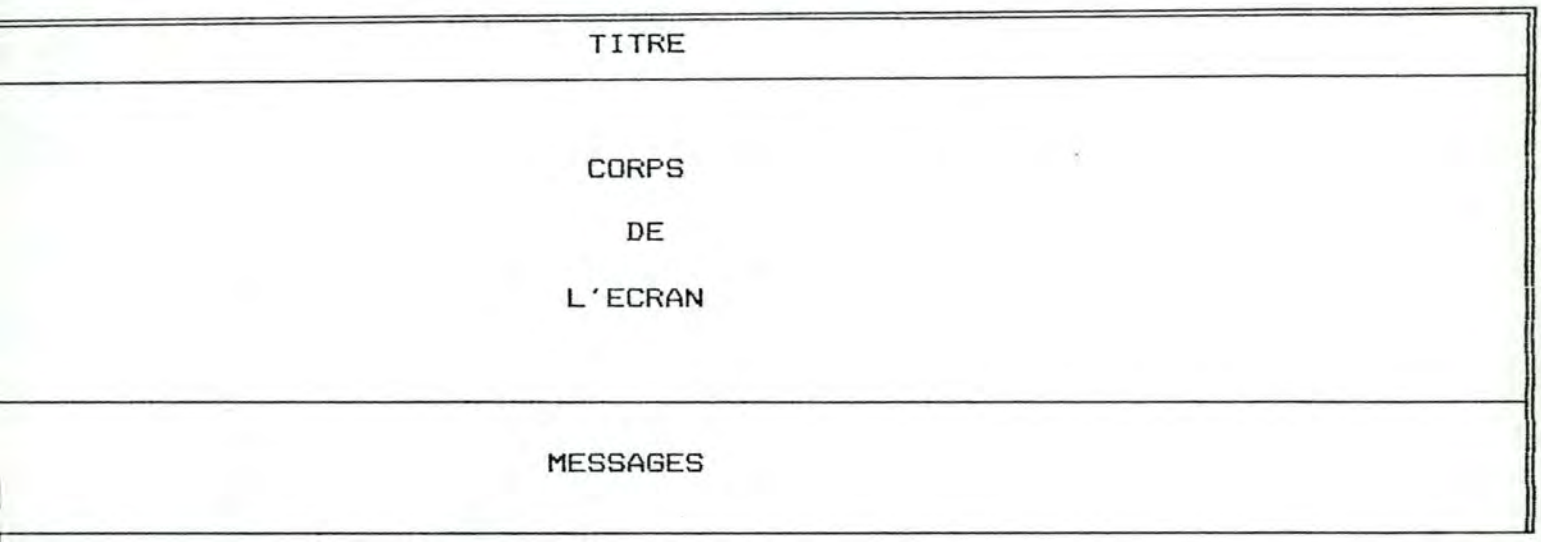
Le sous-ensemble MESSAGES :

> Il est destiné à recevoir le texte des messages : erreurs, invitations,...

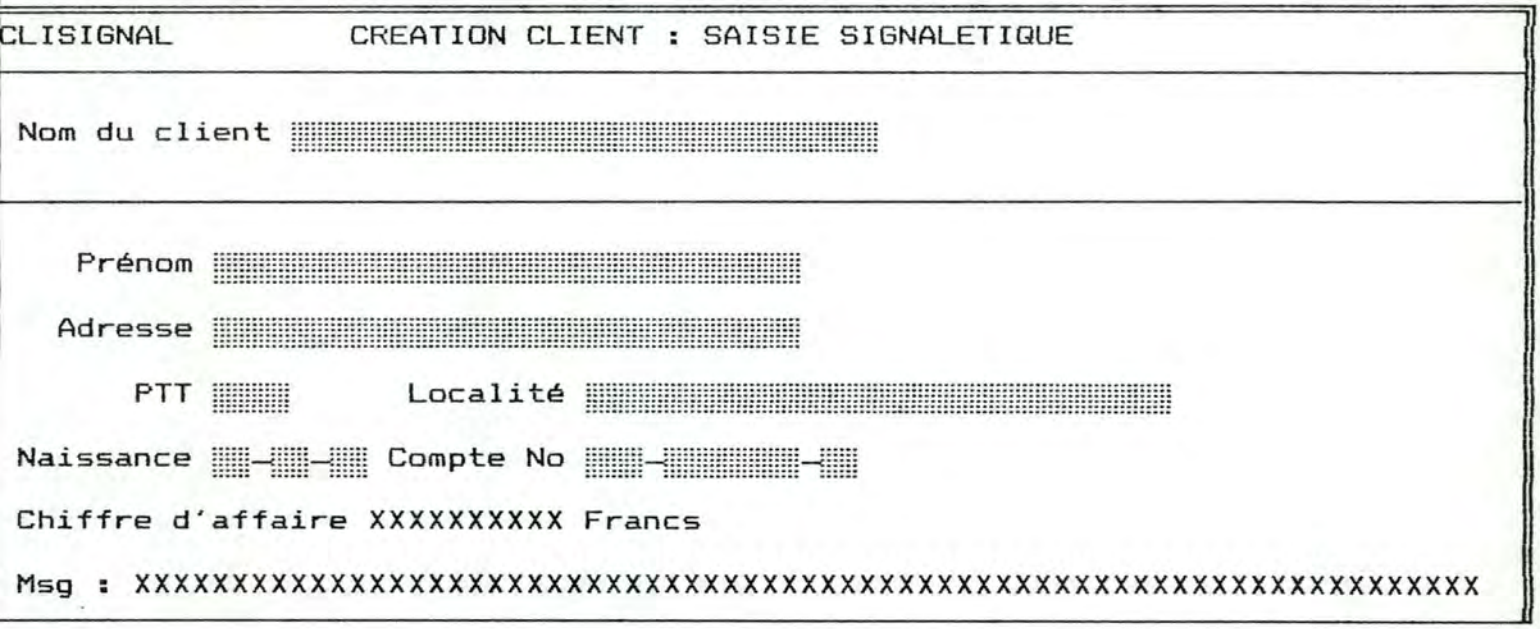
Le sous-ensemble OPTIONS :

> Il informe l'utilisateur des choix qu'il peut effectuer à partir de l'écran affiché. En pratique les options sont basées sur l'utilisation des touches de fonction.

IGURE 2.1. Position géométrique des sous-ensembles d'un écran.



IGURE 2.2. Un exemple d'image-écran



F10 = Acceptation et création Esc = Retour menu

2.2 Constituants des images-écrans.

De ce qui précède, on peut dégager les constituants de base des images-écrans. Nous allons utiliser un exemple formé d'un écran qui permet à l'utilisateur d'encoder le signalétique d'un client (Voir figure 2.2.).

Ces constituants de base sont :

- Un ensemble ordonné de caractères formant le texte de l' image-écran, ce texte est figé par le développeur et non accessible à l'utilisateur. Un texte est lié à un sous-ensemble de l'écran. Dans l'exemple, le texte "SAISIE SIGNALETIQUE" appartient au sous-ensemble Titre et le texte "Nom du client" appartient au sous-ensemble Corps d'écran.

- Un ensemble de portions de l'écran réservées à l'affichage des caractères tapés par l'utilisateur et appelées zones de saisie. Elles sont représentées sur la figure 2 par le caractère spécial '▒'.

- Un ensemble de portions de l'écran réservées à l'affichage de texte évoluant au cours du déroulement de l'application, ces portions sont appelées zones de sortie (Heure, messages divers, ..). Sur la figure 2, ces zones sont représentées par le caractère 'X'.

- Un nom qui permet d'identifier un écran parmi l'ensemble des écrans utilisés par un programme. L'image-écran de la figure 2.2. s'appelle "Clisignal".

Lorsque cet écran est proposé à l'utilisateur, ce dernier y édite les informations, l'image-écran pourra alors avoir l'apparence de la figure 2.3.

IGURE 2.3. Un exemple d'image-écran éditée par l'utilisateur

```
CLISIGNAL          CREATION CLIENT : SAISIE SIGNALETIQUE

 Nom du client DUPONT

    Prénom Henri
   Adresse rue du préau, 67
       PTT 6543  Localité Bruxelles-lez-antoing
 Naissance 26-02-59 Compte No 012-7896543-12
 Chiffre d'affaire          0.00 Francs
 Msg : Le client DUPONT a été enregistré avec succès.

F10 = Acceptation et création                    Esc = Retour menu
```

2.3 Les différents types d'écrans.

2.4 Les écrans de dialogue.

Les écrans de dialogue sont nécessaires à l'affichage et à la saisie des données. La figure 2.2. représente un écran de ce type.

2.5 Les écrans de menu .

Un écran de menu propose à l'utilisateur un ensemble de choix, c'est-à-dire l'ensemble des possibilités offertes par l'application à un moment donné.
On peut apparenter un menu à un aiguillage dans les traitements.
Un écran de menu est illustré à la figure 2.4.

2.6 Les écrans de servitude.

De manière à lui faciliter la tâche il est souvent utile de renseigner l'utilisateur sur la conduite à tenir face à un problème.
Le "guide opérateur ou écran d'aide" est l'exemple type d'écran de servitude. L'écran de servitude ne modifie pas le déroulement du dialogue mais fournit des informations supplémentaires sur le fonctionnement du programme ou sur l'écran de dialogue affiché.
La figure 2.5. illustre un écran de servitude.

2.7 Les attributs d'affichage.

Les attributs d'affichage sont dépendants des caractéristiques du terminal et permettent d'attirer l'attention de l'utilisateur sur un texte affiché sur l'écran, ou de distinguer les textes et les zones de saisie.

IGURE 2.4. Un exemple d'écran de menu

```
                    SELECTION DU FICHIER

                1.  Gestion du fichier des clients
                2.  Gestion du fichier des articles

                    Entrez votre choix : D
```

F10 = Acceptation Esc = Fin du programme

IGURE 2.5. Un exemple d'écran de servitude

```
Ecran d'aide de l'écran CLISIGNAL.

Le nom du client constitue la clef d'accès au client, veillez donc à bien
l'orthographier de manière à éviter des problèmes pour les accès futurs.

Lorsque vous avez terminé l'édition du signalétique, pressez sur la touche
F10 pour enregistrer ces informations. En cas d'erreur d'introduction, pres-
sez sur la touche Esc pour ne pas enregistrer d'informations erronées.

Pressez sur une touche pour revenir à l'écran CLISIGNAL.
```

2.8 Les modes d'édition d'un écran de dialogue.

Lorsqu'un écran de dialogue est affiché, l'utilisateur peut éditer les zones de saisie. Deux techniques sont envisageables :

- L'utilisateur valide chaque édition de zone de saisie par une touche spéciale (par exemple "Enter"). Dans ce cas, une zone de saisie n'est accessible qu'à partir de la zone précédemment éditée. Il s'agit du mode "zone"

- L'utilisateur édite les zones de saisie de l'écran et valide l'ensemble de l'écran à l'aide d'une touche spéciale (par exemple "Send" ou "F10"). Une zone de saisie est alors accessible à partir de n'importe quelle autre à l'aide de touches spéciales (par exemple les touches de déplacement du curseur). Il s'agit du mode "écran".

Il faut remarquer que le choix du mode d'édition d'un écran de dialogue peut dépendre des caractéristiques des terminaux utilisés. Les performances d'un terminal synchrone sont nettement supérieures en mode écran qu'en mode zone. En effet, le mode écran sur un terminal "intelligent" permet l'édition complète de l'écran sans nécessiter d'autres ressources que le terminal lui-même. L'acceptation d'un écran provoque alors la transmission vers l'ordinateur de l'entièreté des informations entrées, en un seul bloc. Il en résulte un temps d'attente nul durant l'édition d'un écran.

2.9 Exemple d'application interactive.

Nous allons utiliser un exemple pour introduire les autres concepts liés à la gestion d'écrans.

Imaginons avoir à programmer une petite application permettant la gestion d'un fichier de clients et d'un fichier d'articles. Les fonctionnalités demandées au programme sont :

- La possibilité de créer, de consulter et de modifier des enregistrements dans chacun des fichiers.
- Lors d'une opération de consultation d'un enregistrement, on voudrait pouvoir consulter les enregistrements précédents et les enregistrements suivants.

Une première solution possible à ce problème est de présenter à l'utilisateur plusieurs menus permettant d'une part la sélection du fichier à utiliser (client ou fournisseur), d'autre part la sélection du type d'opération à effectuer sur le fichier sélectionné (consulter, créer ou modifier). Une fois ces choix effectués, le programme doit demander à l'utilisateur la clef d'accès à l'enregistrement de façon à permettre à celui-ci de mener à bien l'opération choisie.

Une seconde solution possible consiste en la suppression du second menu de la première solution. Les diverses opérations s'effectueraient alors par l'utilisation de touches de fonction. L'utilisateur validerait alors la clef d'accès par une touche indiquant l'opération qu'il veut effectuer (Figure 2.7.).

2.10 Images-écrans induites par l'exemple.

En se basant sur les deux exemples ci-avant, on peut déduire les images écran nécessaires à la mise en oeuvre, celles-ci sont représentées aux figures 2.6. et 2.7. respectivement pour la première et la seconde solution. Ces images représentent l'aspect du terminal tel qu'il serait vu par un utilisateur. Les zones contenant les caractères '|||', 'x' et 'd' apparaissant en écriture renforcée représentent les zones de saisie. Ces images font partie des listings imprimés par le programme PRT qui est décrit au paragraphe 3.21.

2.11 Analogie entre dialogue-écran et graphe.

On peut schématiser les possibilités offertes par le programme à l'aide des images-écrans définies et des enchaînements possibles de ces images.
Dans un premier temps, nous allons représenter les écrans par un rectangle et les enchaînements possibles par des flèches. L'orientation des flèches indique quel écran est "le père" et quel écran est "le fils".

Les figures 2.8. et 2.9. donnent respectivement une représentation possible pour la première et la seconde solution.

Nous pouvons établir un parallèle entre ces deux schémas et des graphes orientés (Ensemble de sommets, les écrans, reliés par des arcs orientés).

Les sommets représentent les états du graphe, les arcs permettent les changements d'états. Dans un dialogue-écran, les états correspondent aux images-écrans et les arcs représentent les procédures de changement. J'appellerai ces procédures les "fonctions-arcs".

IGURE 2.6. Images-écrans de la premiere solution.

SELECTION DU FICHIER

1. Gestion du fichier des clients
2. Gestion du fichier des articles

Entrez votre choix : **D**

F10 = Acceptation Esc = Fin du programme

SELECTION DE L'OPERATION SUR LE FICHIER CLIENTS

1. Consultation d'un client
2. Création d'un client
3. Modification d'un client
4. Suppression d'un client

Entrez votre choix : **D**

F10 = Acceptation du choix Esc = Retour menu

SELECTION DE L'OPERATION SUR LE FICHIER ARTICLES

1. Consultation d'un article
2. Création d'un article
3. Modification d'un article
4. Suppression d'un article

Entrez votre choix : **D**

F10 = Acceptation du choix Esc = Retour menu

CLIENTS : SAISIE DE LA CLEF D'ACCES

Nom du client xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

F10 = Acceptation de la clef Esc = Retour menu

ARTICLES : SAISIE DE LA CLEF D'ACCES

No de l'article xxxxxxxxxxxxxx

F10 = Acceptation de la clef   Esc = Retour menu

CREATION CLIENT : SAISIE SIGNALETIQUE

Nom du client

Prénom xxxxxxxxxxxxxxxxxxx

Adresse xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

PTT **dddd** Localité xxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Naissance **dd-dd-dd** Compte No **dd-ddddddd-ddd**

F10 = Acceptation et création   Esc = Retour menu

MODIFICATION CLIENT : SAISIE SIGNALETIQUE

Nom du client

Prénom xxxxxxxxxxxxxxxxxxxx

Adresse xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

PTT **dddd** Localité xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Naissance **dd-dd-dd** Compte No **dd-ddddddd-ddd**

F10 = Acceptation et modification Esc = Retour menu

CONSULTATION CLIENT : SAISIE SIGNALETIQUE

Nom du client

Prénom

Adresse

PTT Localité

Naissance Compte No

PgUp = Client précédent PgDn = Client suivant F10 = Esc = Retour menu

CREATION ARTICLES : SAISIE SIGNALETIQUE

No de l'article

Désignation xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Omschrijving xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Taux TVA.

Prix achat **dddddddd** Prix vente HTVA **dddddddd** % TVA **dd**

Qté en stock **dddd** Qté en commande fournisseur **dddd**
client **dddd**

F10 = Acceptation et création Esc = Retour menu

MODIFICATION ARTICLES : SAISIE SIGNALETIQUE

No de l'article

Désignation xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Omschrijving xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Taux TVA.

Prix achat **dddddddd** Prix vente HTVA **dddddddd** % TVA **dd**

Qté en stock **dddd** Qté en commande fournisseur **dddd**
client **dddd**

F10 = Acceptation et modification Esc = Retour menu

CONSULTATION ARTICLES : SAISIE SIGNALETIQUE

No de l'article

Désignation
Omschrijving

Taux
TVA.

Prix achat Prix vente HTVA % TVA

Qté en stock Qté en commande fournisseur
client

PgUp = Article précédent PgDn = Article suivant F10 = Esc = Retour menu

GURE 2.7. Images-écrans de la seconde solution.

SELECTION DU FICHIER

F10 = Acceptation

Esc = Fin du programme.

1. Gestion du fichier des clients

2. Gestion du fichier des articles

Entrez votre choix : **D**

CLIENT : SAISIE DE LA CLE D'ACCES

F10= Consultation

F2 = Création

F3 = Modification

F7 = Suppression

PgUp=Client préc.
PgDn=Client suiv.

Esc = Retour menu

Nom du client xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Prénom

Adresse

PTT Localité

Date de naissance Compte No

ARTICLES : SAISIE DE LA CLES D'ACCES

No de l'article xxxxxxxxxxxxxxx

Désignation
Omschrijving

Prix achat Prix vente HTVA % TVA

Qté en stock Qté en commande fournisseur
client

Taux de TVA valides :

F10= Consultation

F2 = Création

F3 = Modification

F7 = Suppression

PgUp=Article préc.
PgDn=Article suiv.

Esc = Retour menu

CLIENT : MODIFICATION DU SIGNALETIQUE

Nom du client

Prénom xxxxxxxxxxxxxxxxxx

Adresse xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

PTT **dddd** Localité xxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Date de naissance **dd-dd-dd** Compte No **dd-ddddddd-ddd**

F10 = Modification et retour sélection.

Esc = Abandon de la modif.

CLIENT : CREATION D'UN CLIENT

Nom du client ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░

Prénom xxxxxxxxxxxxxxxxxxxx

Adresse xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

PTT **dddd** Localité xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Date de naissance **dd-dd-dd** Compte No **dd-ddddddd-ddd**

F10 = Création et retour sélection.

Esc = Abandon de la création

ARTICLES : MODIFICATION DU SIGNALETIQUE

No de l'article ░░░░░░░░░░░░░░░

Désignation xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Omschrijving xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Prix achat **dddddddd** Prix vente HTVA **dddddddd** % TVA **dd**

Qté en stock **dddd** Qté en commande fournisseur **dddd**
client **dddd**

Taux de TVA valides : ░░ ░░ ░░ ░░ ░░ ░░ ░░

F10= Modification et retour sélection.

Esc = Abandon de la modif.

ARTICLES : CREATION D'UN ARTICLE

No de l'article ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Désignation xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Omschrijving xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Prix achat **dddddddd** Prix vente HTVA **dddddddd** % TVA **dd**

Qté en stock **dddd** Qté en commande fournisseur **dddd**
client **dddd**

Taux de TVA valides : ▒▒ ▒▒ ▒▒ ▒▒ ▒▒ ▒▒ ▒▒

F10= Création et retour sélection.

Esc = Abandon de la création

FIGURE 2.8. Graphe de la première solution.

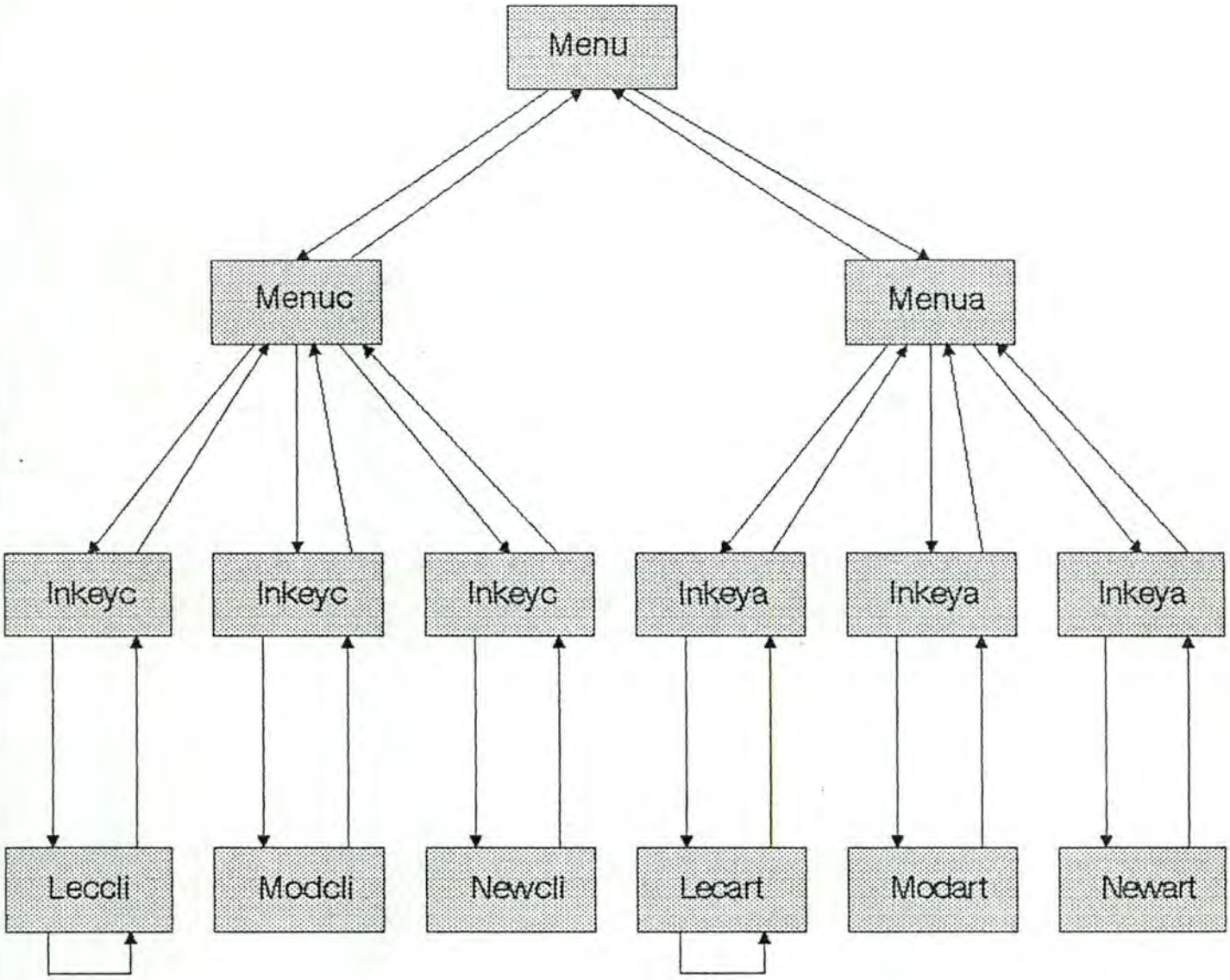
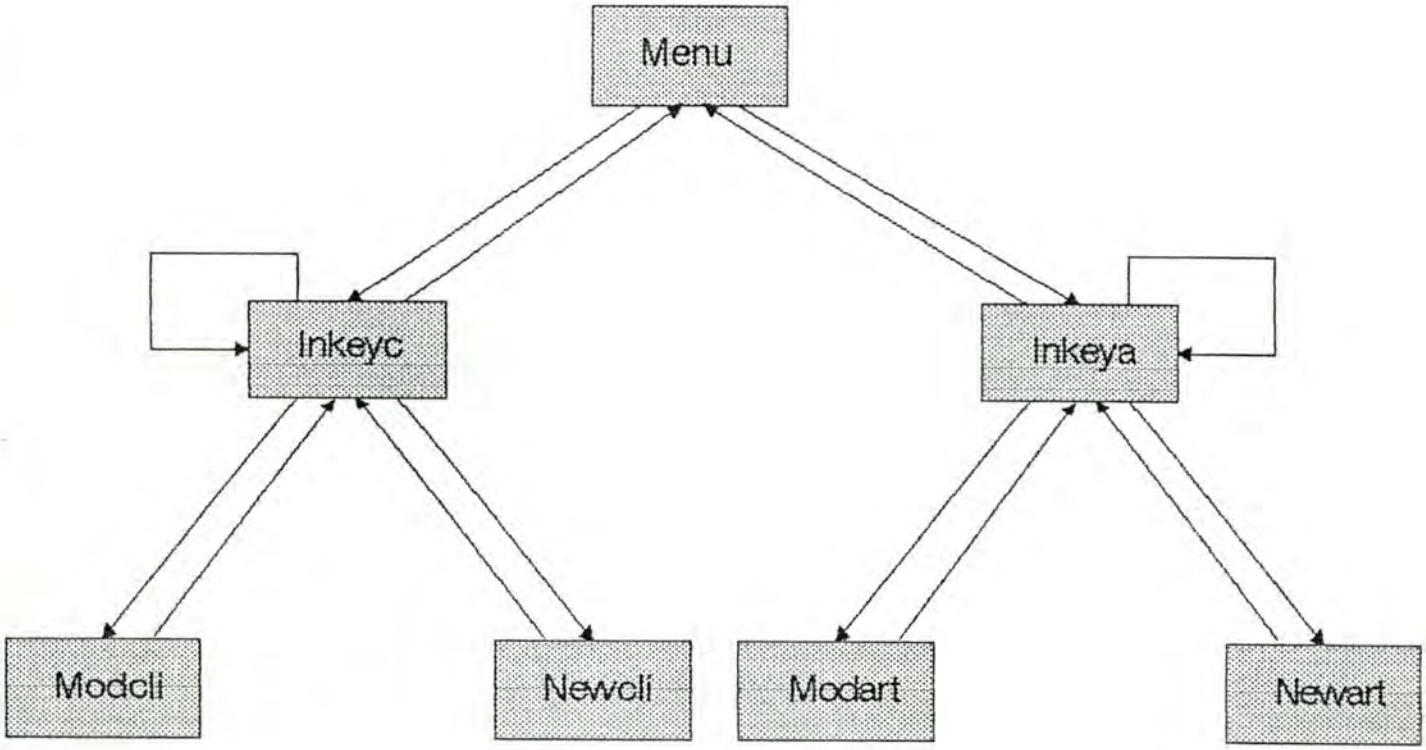


FIGURE 2.9. Graphe de la seconde solution.



2.12 Les touches de fonction.

Nous avons déjà évoqué les touches de fonction, il faut maintenant en préciser l'utilisation. Comme nous l'avons vu dans la seconde solution, ces touches permettent de choisir une opération à effectuer. Pour conserver la plus grande convivialité possible il faut, dans chaque application, attribuer à chaque touche de fonction une fonction similaire. Pour ce faire, nous avons mené une brève étude sur des logiciels existants. Il s'en dégage une standardisation concernant un nombre très restreint de touches de fonction et qui attribue à ces dernières un rôle déterminé. Voici les différentes observations que nous avons faites :

F1 : Touche d'aide.

F10: Touche d'acceptation d'écran et avancement dans le logiciel (Parfois appelée SEND ou DO).

Esc: Refus de l'écran affiché et recul dans l'arborescence (Parfois appelée UNDO).

Return ou **Enter**: Acceptation d'une zone de saisie.

Page Up et **Page Down** : Passage à un élément suivant ou précédent, par exemple "client suivant" ou "client précédent" (sens précisé par le contexte).

Touches de **déplacement du curseur** :Elles sont fréquemment utilisées pour se déplacer au sein d'une zone ou d'une zone à une autre.

Insert et **Delete** : Utilisées pour la correction lors de l'édition d'une zone de saisie.

Les touches de fonction restantes sont laissées à la disposition des logiciels, leur utilisation varie d'une application à l'autre.

Il faut insister sur le fait qu'une grande dispersion existe dans l'attribution de fonctions à certaines touches de fonction, cette dispersion est encore amplifiée par la grande variété de terminaux existants.

2.13 Les appels de fonctions-arcs et le scénario.

Dans un graphe, plusieurs arcs peuvent être issus d'un même sommet. L'utilisateur doit pouvoir déterminer quel arc choisir, c'est-à-dire définir les modalités d'appel aux fonctions-arc. Un des moyens couramment utilisé est l'appui sur une touche de fonction.

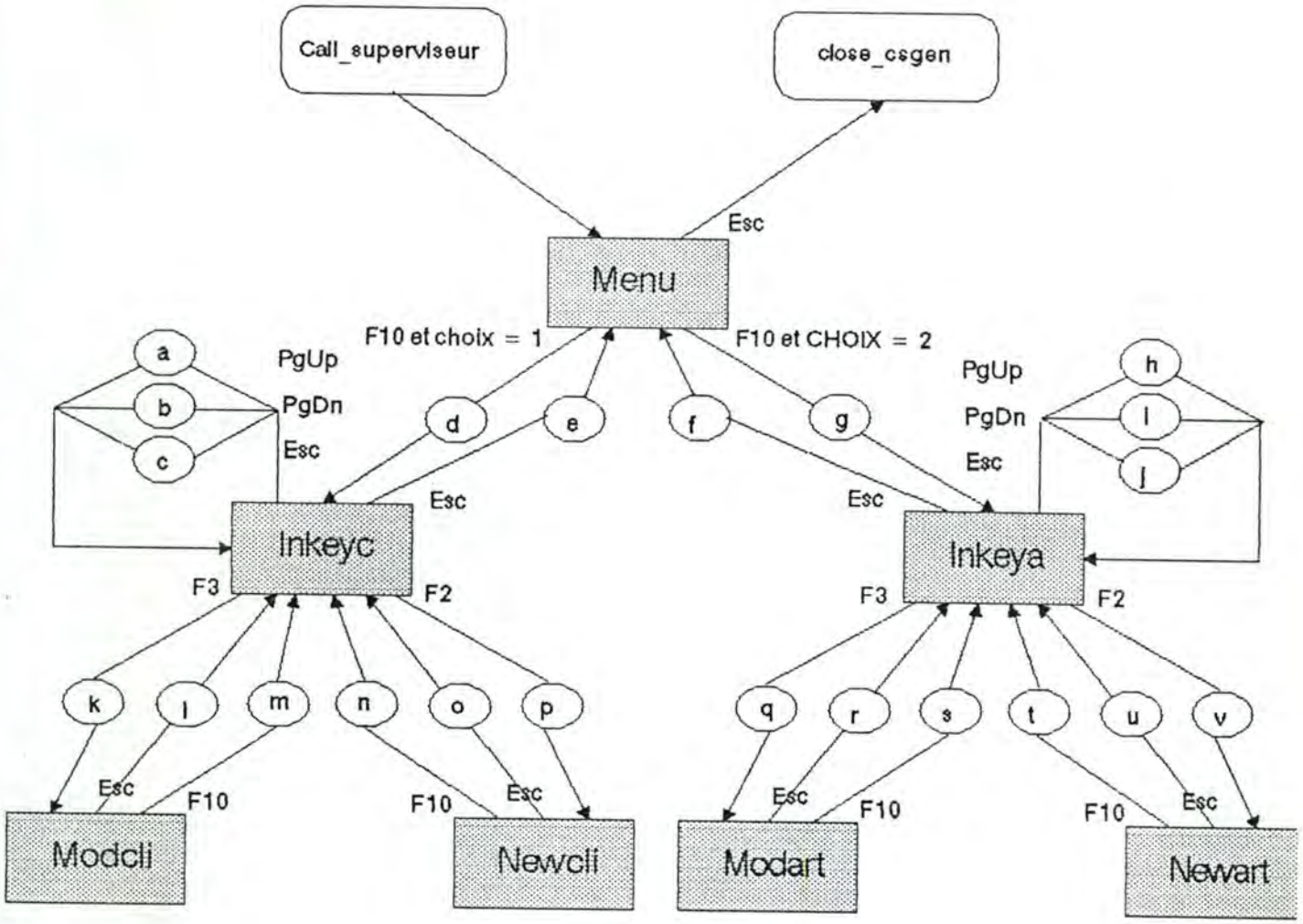
Cependant, les touches de fonctions ne suffisent pas à déterminer complètement les modalités d'appel. Par exemple, un écran de menu est généralement validé par une seule touche de fonction bien que différentes fonctions-arcs y soient attachées. Il faut donc ajouter un test qui, en liaison avec une touche de fonction, déterminera quelle fonction-arc doit être utilisée et donc quel sera le prochain écran affiché.

Une fonction-arc est identifiée par un nom qui doit répondre à la syntaxe admise par le langage C.

En reprenant l'exemple de la seconde solution (Figure 2.8), on peut compléter le schéma du dialogue-écran comme illustré en figure 2.10.

Le schéma ainsi complété permet de lever toute ambiguïté sur l'enchaînement des images-écrans. Ce schéma représente le "scénario" de l'application.

FIGURE 2.10. Graphe complet de la seconde solution.



x = *Fonction – arc*

= *Ecran*

PgUp, PgDn, Esc, F2, F3, Esc = Touches de fonction

2.14 Les écrans logiques et physiques.

Les images-écrans abordées jusqu'à présent représentent les images telles qu'elles sont affichées sur l'écran du terminal, il s'agit d'écrans physiques.

Cependant, on remarque par exemple que l'image de l'écran Inkeyc se retrouve en partie dans l'image de l'écran Modcli, ce qui implique qu'il n'est pas nécessaire d'afficher l'entièreté de l'écran Modcli pour passer de l'écran Inkeyc à l'écran Modcli. On divise dés lors l'écran Modcli en deux sous-ensembles, ce sont les écrans logiques constituant l'écran physique Modcli.
Bien entendu, il ne peut être question de changer le numéro d'article de l'écran Modcli, celui-ci étant uniquement affiché pour mémoire. L' écran logique constituant l'image-écran Inkeyc à donc même apparence dans les deux images-écrans, mais la zone de saisie de l'écran Inkeyc est transformée en zone de sortie dans l'écran Modcli.

Les écrans ci-dessous représentent les écrans logiques relatifs à la seconde solution. On remarquera que le dessin d'un écran logique dépend fortement du scénario qui a été défini au paragraphe 2.13.

om de l'écran : modcli

CLIENT : MODIFICATION DU SIGNALETIQUE	
xxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx dddd xxxxxxxxxxxxxxxxxxxxxxxxxxxxx dd_dd_dd dd_ddddddd_ddd	F10 = Modification et retour sélection. Esc = Abandon de la modif.

m de l'écran : newcli

CLIENT : CREATION D'UN CLIENT	
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx dddd xxxxxxxxxxxxxxxxxxxxxxxxxxxxx dd_dd_dd dd_ddddddd_ddd	F10 = Création et retour sélection. Esc = Abandon de la création

om de l'écran : inkeya

```
                    ARTICLES : SAISIE DE LA CLES D'ACCES

  o de l'article xxxxxxxxxxxxxxx                    F10= Consultation

  ignation                                          F2 = Création
  chrijving                                         F3 = Modification

                                                    F7 = Suppression
  Prix achat          Prix vente HTVA          % TVA

                                                    PgUp=Article préc.
  Qté en stock        Qté en commande fournisseur   PgDn=Article suiv.
                                        client

  Taux de TVA valides :                             Esc = Retour menu
```

om de l'écran : modart

```
                ARTICLES : MODIFICATION DU SIGNALETIQUE

                                                    F10= Modification
                                                         et retour
                                                         sélection.
        xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

        dddddddd                dddddddd        dd  Esc = Abandon de
                                                        la modif.
        dddd                                dddd
                                            dddd
```

m de l'écran : newart

ARTICLES : CREATION D'UN ARTICLE

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

dddddddd dddddddd dd

dddd
dddd
dddd

F10= Création
et retour
sélection.

Esc = Abandon de
la création

CHAPITRE 3 : CSGEN UTILITAIRE.

La première version de CSGen a été réalisée sous la forme d'un utilitaire composé de :

- Un éditeur pour la confection d' écrans logiques.
- Un compilateur d'écrans.
- Une librairie de primitives pour exploiter les écrans ainsi conçus.

Les exigences définies dans l'étude des besoins et l'approche "utilitaire" ont mené à l'adoption de certains concepts partiellement abordés au chapitre 2, mais que nous allons maintenant définir plus précisément. Il est a noter que notre prise de conscience de la notion de scénario est largement postérieure au développement de CSGen utilitaire, c'est la raison pour laquelle celui-ci n'y fait pas référence.

Nous allons reprendre, dans l'ordre, les étapes qui ont abouti à la version définitive de CSGen utilitaire.

Une des nécessités impliquées par l'ergonomie peut se traduire par l'adage : "Une place pour chaque chose, chaque chose a sa place."
Intuitivement, on peut donner un "type" à chaque texte affiché à l'écran. Ce "type" est le reflet de la sémantique du message. Par exemple, un message d'erreur ne possède pas la même sémantique qu'une question posée à l'utilisateur.

Durant tout le déroulement d'une application chaque type de message échangé entre l'homme et la machine doit avoir une localisation bien précise sur l'écran du terminal. On peut ainsi "découper" l'écran en zones qui réceptionneront un type de message. Quatre zones sont rapidement perceptibles :

- Une zone "titre" correspondant au titre de l'écran affiché.
- Une zone "corps d'écran" où se déroulera le dialogue homme-machine.
- Une zone "erreur" où seront affichés les messages d'erreur.
- Une zone "option" indiquant à l'utilisateur quelles sont les touches de fonction actives à partir de l'écran affiché.

D'autre zones pouvant se révéler utiles ont été inclues au gestionnaire d'écran :

- Une zone "nom d'écran" facilitant par exemple le debugging par téléphone.
- Une zone "type d'erreur" indiquant la provenance du message d'erreur affiché (SGBD, DISQUE, CLAVIER,...)
- Une zone "message" affichant des ordres à l'utilisateur.
- Une zone "clé accédée" rappelant par exemple le nom d'un client lors de la rédaction d'une facture.

3.1 L'écran de fond.

Pour permettre la définition de ces zones, nous avons introduit la notion d'écran "de fond". Chaque écran logique sera réalisé en fonction de l'écran de fond défini par le développeur.

Outre cette définition, l'écran de fond permet de séparer visuellement les zones en les plaçant par exemple dans des cadres, de manière à en améliorer l'esthétique tout en apportant un confort d'utilisation accru. L'écran de fond permet également, pour chacune de ses zones, de définir les attributs d'affichage des textes. En fait, CSGen permet pour chaque zone d'utiliser une conjonction de trois attributs choisis parmi ceux-ci :

- Affichage normal.
- Surbrillance.
- Contraste inversé.
- Clignotement.
- Soulignage.
- Une palette de seize couleurs de fond de caractères.
- Une palette de seize couleurs de caractères.

3.2 Composantes des écrans logiques.

L'étape suivante fut de définir les fonctionnalités du gestionnaire d'écran au niveau de la confection des écrans, de l'exploitation de ceux-ci ainsi qu'au niveau de l'utilisateur final.

Nous avons considéré un écran logique comme étant constitué d'un ensemble de caractères affichés à l'écran et d'un ensemble de zones dites "de saisie" dans lesquelles l'utilisateur éditera les informations (les zones de saisie constituent les seules portions de l'écran éditables par l'utilisateur final).

Un écran logique est donc actuellement constitué des éléments suivants :

- Des caractères dans la zone "titre"
- Des caractères dans la zone "option"
- Des caractères dans la zone "corps"
- Des zones de saisie dans la zone "corps"

3.3 Liaison écran - programme.

Il fallait ensuite choisir le mode de liaison entre une zone de saisie et un objet manipulable par le langage C et donc par le programmeur. Le moyen le plus naturel consistait à associer une zone de saisie à une variable du programme.

Pour garder suffisamment de souplesse d'emploi nous avons choisi de n'utiliser qu'un seul des types de variables admis par le langage C : Les chaînes de caractères ou vecteurs de caractères, ceux-ci pouvant être utilisés sous n'importe quelle forme : simple vecteur ("string"), tableau multidimensionnel ou membre d'une structure. En langage C, une chaîne de caractères est une zone mémoire destinée à recevoir des données de type caractères de 8 bits chacune; la fin de la chaîne étant par convention le caractère nul (codé comme un zéro binaire). Le fait de n'utiliser que des chaînes de caractères ne constitue pas une grande limitation puisque le langage C fournit la plupart des fonctions de conversion entre les différents types.

3.4 Type des zones de saisie.

Il fallait également définir le "type" des zones de saisie, c'est-à-dire informer le gestionnaire d'écrans de la possibilité ou de l'impossibilité pour l'utilisateur de modifier une zone de saisie affichée. J'ai utilisé trois types de zones de saisie :

- Output : La zone n'est pas modifiable par l'utilisateur (elle équivaut dans ce cas à une zone de sortie).
- Modif : La zone est modifiable par l'utilisateur.
- Input : La zone doit être initialisée avant d'être modifiée.

Pour plus de clarté dans la suite, nous parlerons de zones de saisies accessibles pour désigner les zones de saisies de type Modif et input et zones de saisies inaccessibles dans le cas de zones de saisie de type output.

La saisie d'un écran peut être vue comme la saisie d'une ou plusieurs zones de saisie. Lorsque plusieurs zones de saisies sont accessibles dans un même écran, on parlera de la zone de saisie courante pour désigner la zone qui est en train d'être éditée. Nous verrons au paragraphe 3.14 les possibilités de changement de zone de saisie courante.

Enfin, un "curseur", représenté sur l'écran par un pavé clignotant, indique l'endroit où le prochain caractère tapé par l'utilisateur sera affiché.

3.5 Les tests de validité.

Un gestionnaire d'écran doit posséder des moyens permettant de tester la validité des informations entrées par l'utilisateur.
Afin d'éviter au programmeur l'apprentissage d'un langage spécifique aux tests de validité de CSGen, nous avons choisi d'utiliser la syntaxe du langage C pour réaliser ces tests. (Des fonctions spécifiques enrichissent les possibilités de test).
La grande liberté syntaxique du langage C permet d'écrire des tests très puissants et très complets, même en limitant le nombre de tests possibles relatifs à une zone de saisie.

Un test de validité est bien entendu lié à un message d'erreur qui informera l'utilisateur d'une erreur d'introduction.

3.6 Quand tester la validité d'une zone de saisie ?

Il existe deux moments dans l'édition d'un écran pour tester la validité des zones de saisie éditées :

- Tester la validité d'une zone après sa saisie.
- Tester la validité de toutes les zones de saisie après acceptation de l'écran affiché.

Ces deux techniques découlent des deux types d'exploitation d'un écran : Le mode zone et le mode écran précédemment définis au paragraphe 2.8. Bien que des variantes soient possibles, en mode zone le test de validité est mené après l'édition d'une zone de saisie alors qu'il est mené pour l'ensemble des zones après l'édition de tout l'écran en mode écran.

Lorsqu'une zone est invalide, un message d'erreur est affiché et la zone fautive est à nouveau soumise à l'édition par l'utilisateur. En mode écran, la réédition des zones éronnées est constituée de l'édition d'un sous-ensemble des zones de saisie accessibles de l'écran.

3.7 Le format d'une zone de saisie

Il peut être intéressant d'effectuer un contrôle des caractères entrés par l'utilisateur tout au long de l'édition d'une zone de saisie de manière à refuser d'emblée un caractère fautif. Pour mener cette tâche à bien, un "format" est associé à chaque zone de saisie accessible.

Le format indique, pour chaque caractère de la zone, l'ensemble des touches admises (approche semblable au "PICTURE" du COBOL). Cette relation biunivoque permet au format d'indiquer également le nombre de caractères acceptés par une zone de saisie.

Dans un format, des lettres représentent le type de caractère accepté à l'emplacement du curseur dans la zone de saisie.

Les zones String (cadrées à gauche)

La lettre "a" représente une position alphabétique.
La lettre "x" représente une position alphanumérique.
La lettre "d" représente une position numérique.

Les zones numériques

Le caractère "." définit la position du point décimal et force la zone de saisie à être numérique, ce qui se traduit par une édition différente d'une zone de type string. En effet, la partie entière du nombre est cadrée à droite alors que la partie décimale est cadrée à gauche du point décimal.

Les caractères d'insertion

Dans un format, un caractère différent des quatres précédent est considéré comme un "caractère d'insertion". Il sera visualisé sur l'écran et inséré dans la variable liée à la zone de saisie mais ne sera pas éditable par l'utilisateur. Les caractères d'insertion permettent d'obtenir une édition plus lisible, par exemple dans le cas de dates ou de numéros de comptes bancaires.

Une date et un compte bancaire pourront avoir respectivement le format suivant : "dd/dd/dd" et "ddd-ddddddd-dd".

3.8 Les touches de sortie d'écran.

Si on envisage l'affichage d'un écran comme consécutif à l'appel à une fonction, il faut que l'exécution de cette fonction se termine de manière à rendre le contrôle au programme appelant. Si la fonction a pour but d'afficher et d'éditer un écran, la terminaison de cette fonction peut être appelée la "sortie d'un écran".
La sortie d'un écran est provoquée par l'appui sur une touche de fonction. Le paragraphe précédent suggère que des touches telles que F10, Esc, PgU, PgD, F2 à F9 sont des touches de sortie possibles, en notant que la touche Esc signifie un refus de l'écran.

CSGen permet de définir les touches permettant de sortir d'un écran, les autres seront ainsi sans effet et le programmeur ne devra pas s'en préoccuper. En effet, lors de chaque appel de fonction impliquant l'édition d'un écran, CSGen renvoie au programme appelant un code représentant la touche de sortie d'écran que l'utilisateur a tapé.

3.9 synthèse.

Compte tenu de ce qui précède, on peut énumérer les éléments constitutifs d'un écran :

- Des caractères dans la zone "titre"
- Des caractères dans la zone "option"
- Des caractères dans la zone "corps"
- Des valeurs représentants les touches de sortie.
- Des zones de saisie dans la zone "corps"

Pour chacune des zones de saisie :

- Le nom de la variable C associée.
- Le type de la zone (In, Mod, Out).
- Le test de validité écrit en C et son message d'erreur.
- Le format et la longueur de la zone (axd.).

Le listing figurant ci-après illustre cette synthèse en l'appliquant à la seconde solution. Ce texte est en fait la sortie imprimée du programme PRT décrit au paragraphe 3.21.

m de variable	ligne	colonne	longueur	attribut
	Test d'invalidité Message d'erreur correspondant			
m de l'écran : menu				
oix	13	37	1	Zone de modification
	if(@[0]!='1'&&@[0]!='2') erreur			
	Entrez soit 1, soit 2 comme choix.			
m de l'écran : inkeyc				
i.nom	4	19	30	Zone de modification
	if(isempty(upper(ttrim(@)))) erreur			
	Le nom du client ne peut être vide.			
i.prenom	7	9	20	Zone de sortie
	if() erreur			
i.adresse	11	10	36	Zone de sortie
	if() erreur			
i.ptt	13	6	4	Zone de sortie
	if() erreur			
i.localite	13	21	30	Zone de sortie
	if() erreur			
i.naiss	17	20	8	Zone de sortie
	if() erreur			
i.banque	17	39	14	Zone de sortie
	if() erreur			

n de variable	ligne	colonne	longueur	attribut
Test d'invalidité Message d'erreur correspondant				

n de l'écran : inkeya

t.num	4	21	15	Zone de modification
if(isempty(upper(ttrim(@)))) erreur Le numéro de l'article ne peut être vide.				
t.design	7	16	40	Zone de sortie
if() erreur				
t.omschr	8	16	40	Zone de sortie
if() erreur				
t.achat	11	13	8	Zone de sortie
if() erreur				
t.vente	11	39	8	Zone de sortie
if() erreur				
t.tva	11	55	2	Zone de sortie
if() erreur				
t.qstock	14	15	4	Zone de sortie
if() erreur				
t.qfourn	14	51	4	Zone de sortie
if() erreur				
t.qclient	15	51	4	Zone de sortie
if() erreur				
a[0]	18	25	2	Zone de sortie
if() erreur				
a[1]	18	30	2	Zone de sortie
if() erreur				

m de variable	ligne	colonne	longueur	attribut
Test d'invalidité Message d'erreur correspondant				
a[2]	18	35	2	Zone de sortie
if() erreur				
a[3]	18	40	2	Zone de sortie
if() erreur				
a[4]	18	45	2	Zone de sortie
if() erreur				
a[5]	18	50	2	Zone de sortie
if() erreur				
a[6]	18	55	2	Zone de sortie
if() erreur				

m de l'écran : modcli

i.prenom	7	9	20	Zone de modification
if() erreur				
i.adresse	11	10	36	Zone de modification
if() erreur				
i.ptt	13	6	4	Zone de modification
if(!inrange(@,1000,9999)) erreur				
Le code postal doit être compris entre 1000 et 9999				
i.localite	13	21	30	Zone de modification
if() erreur				
i.naiss	17	20	8	Zone de modification
if(!isdate(@)) erreur				
La date de naissance est invalide (JJ-MM-AA).				

```
 de variable        ligne     colonne   longueur  attribut
       Test d'invalidité
       Message d'erreur correspondant
-------------------------------------------------------------------------------

.banque              17        39        14        Zone de modification
       if(!isbank(@)) erreur
       Le numéro de compte en banque est invalide.
 de l'écran : newcli
.nom                 4         19        30        Zone de modification
       if(isempty(ttrim(upper(@)))) erreur
       Le nom du client ne peut être vide.
.prenom              7         9         20        Zone d'entrée
       if() erreur
.adresse             11        10        36        Zone d'entrée
       if() erreur
.ptt                 13        6         4         Zone d'entrée
       if(!inrange(@,1000,9999)) erreur
       Le code postal doit être compris entre 1000 et 9999
.localite            13        21        30        Zone d'entrée
       if() erreur
.naiss               17        20        8         Zone d'entrée
       if(!isdate(@)) erreur
       La date de naissance est invalide (JJ-MM-AA).
.banque              17        39        14        Zone d'entrée
       if(!isbank(@)) erreur
       Le numéro de compte en banque est invalide.
```

n de variable	ligne	colonne	longueur	attribut
Test d'invalidité Message d'erreur correspondant				

n de l'écran : modart

t.design	7	16	40	Zone de modification
if() erreur				
t.omschr	8	16	40	Zone de modification
if() erreur				
t.achat	11	13	8	Zone de modification
if() erreur				
t.vente	11	39	8	Zone de modification
if(islower(@,art.achat)) erreur				
Le prix de vente ne peut être inférieur au prix d'achat.				
t.tva	11	55	2	Zone de modification
if(!istva(@)) erreur				
Le pourcentage de TVA est invalide.				
t.qstock	14	15	4	Zone de modification
if() erreur				
t.qfourn	14	51	4	Zone de modification
if() erreur				
t.qclient	15	51	4	Zone de modification
if() erreur				

m de variable	ligne	colonne	longueur	attribut
	Test d'invalidité Message d'erreur correspondant			
m de l'écran : newart				
t.num	4	21	15	Zone de sortie
	if() erreur			
t.design	7	16	40	Zone d'entrée
	if() erreur			
t.omschr	8	16	40	Zone d'entrée
	if() erreur			
t.achat	11	13	8	Zone d'entrée
	if() erreur			
t.vente	11	39	8	Zone d'entrée
	if(islower(@,art.achat)) erreur			
	Le prix de vente ne peut être inférieur au prix d'achat.			
t.tva	11	55	2	Zone d'entrée
	if(!istva(@)) erreur			
	Le pourcentage de TVA est invalide.			
t.qstock	14	15	4	Zone d'entrée
	if() erreur			
t.qfourn	14	51	4	Zone d'entrée
	if() erreur			
t.qclient	15	51	4	Zone d'entrée
	if() erreur			

3.10 Génération de code source.

Sachant que les variables utilisées par les écrans sont de type "chaîne de caractères" et connaissant leur nom et leur longueur, on dispose de toutes les informations nécessaires à la définition des variables au sens C du terme. C'est pourquoi CSGen génère un fichier de définition des variables qui peut être inclus dans le code source du programme, évitant ainsi ce travail d' encodage.

La même approche est utilisée pour utiliser les tests de validité associés aux zones de saisie. Ce procédé sera décrit en détail au paragraphe 3.20.

3.11 Pourquoi une pile d'écrans ?

Jusqu'a présent, nous avons exposé un écran comme formant une entité entièrement affichée sur l'écran ou entièrement effacée. En fait, il faut distinguer les notions d'écran physique et d'écran logique telles qu'elles sont définies au paragraphe 2.14.

Il faut maintenant trouver une relation entre les écrans logiques et physiques. On voit qu'un écran physique est constitué de la superposition de plusieurs écrans logiques. Une pression sur la touche F2 , lors de l'édition de l'écran Inkeyc, provoquera l'affichage du second écran logique à savoir Newcli; alors que dans celui-ci, la même pression de touche effacera celui-ci.

Cette constatation a amené à considérer un écran physique comme un empilement d'écrans logiques. On connait les opérateurs de base d'une structure de pile : INITIALISATION, POP, PUSH et TOP. Ici le push d'un écran signifie son affichage; le pop d'un écran revient à effacer celui-ci; le top d'un écran consiste à réafficher l'écran au sommet de la pile.
Il faut maintenant analyser plus en détail les réactions des différentes composantes d'un écran :

PUSH :

- Afficher les zones titre et option.
- Initialiser les zones de saisie déclarées en input.
- Afficher le texte et les zones de saisie du corps de l'écran.
- Permettre l'édition des zones de saisie déclarées en input et modif et interdire l'édition des zones des autres écrans de la pile.

POP :

- Effacer les informations affichées lors du push.
- Restaurer l'affichage des zones titre et option de l'écran qui devient sommet de la pile.
- Restaurer l'affichage du texte de la zone corps de l'écran qui devient sommet de la pile et qui aurait été effacé par l'écran que l'on enlève de la pile. Cela pose un problème car il se

peut qu'il faille restaurer l'affichage du texte de chaque écran de la pile si l'écran enlevé de la pile occupe une surface importante de la zone corps d'écran. Une autre approche est d'interdire la superposition de textes dans la pile, ce qui évite évidemment d'avoir à les restaurer. C'est cette dernière solution qui a été adoptée dans CSGen.
- Pour les mêmes raisons il faudrait restaurer les zones de saisie de tous les écrans appartenant à la pile. Ici le problème dépasse la simple superposition puisque les zones de saisie représentent des variables dont la valeur évolue durant l'exécution du programme. La solution adoptée à été d'obliger le programmeur à définir dans un écran logique toutes les zones dont la valeur peut avoir évolué.
- Il faut ensuite permettre l'édition des zones de saisie déclarées en input et modif de l'écran devenant le sommet de la pile.

TOP :

- Il suffit de restaurer l'affichage des zones de saisie puisque l'image écran reste fixe.

3.12 Les écrans de type menu

Nous avons jusqu'à présent exposé l'écran de fond et des écrans que nous appellerons "de dialogue". A des fins d'ergonomie, les écrans de choix sont considérés comme un type particulier d'écran. Il est en effet courant de trouver des menus dans lesquels le choix du traitement à effectuer s'effectue en déplaçant, à l'aide de touches de fonction, un "repère" en face des différents points du menu; une autre touche permettant alors d'entériner ce choix.

L'absence de zone de saisie et le traitement particulier lié à la gestion du repère nous a obligé à considérer un menu comme un écran de type particulier. La seule valeur retournée au programme par un menu est la valeur associée au point de menu choisi, ou "Esc" en cas de refus du menu.
La valeur associée à un point de menu est définie par l'utilisateur et n'est pas liée à la position de ce point sur l'écran, le programmeur peut ainsi modifier l'apparence d'un menu sans avoir à modifier le programme l'utilisant.

Il faut noter que l'affichage d'un écran de type menu vide automatiquement la pile des écrans puisqu'il représente un aiguillage vers différents traitements donc vers différentes images d'écran. ceci empêche de superposer un menu sur un quelconque écran ce qui signifie dans le cas d'un menu, que l' écran logique est identique à l' écran physique.

3.13 Les écrans d'aide

Toujours dans un but de convivialité, une aide (ou guide) doit être disponible à n'importe quel moment durant le déroulement du programme. Pour cette raison, un écran de servitude peut être défini par le programmeur en liaison avec un écran logique. Un appui sur la touche d'aide affichera l'écran esclave, ensuite, un appui sur une touche quelconque produira le réaffichage de l'écran "maître" tel qu'il se trouvait avant la demande d'aide.

3.14 Les fonctions disponibles lors de la saisie d'un écran.

Il faut distinguer deux concepts lors de la saisie d'un écran :

- La manière d'accéder à une zone de saisie.
- La saisie proprement dite d'une zone de saisie.

Dans ce paragraphe, les termes "zones de saisie" font référence à des "zones de saisie accessibles" telles qu'elles sont définies au paragraphe 3.4.
Dans CSGen, un sous-ensemble de touches de fonction est réservé au "voyage" entre les zones de saisie d'un écran. Selon la disposition géométrique des zones de saisie sur l'écran, la zone de saisie "suivante" sera différente. Cette disposition détermine si l'écran affiché représente ou non un tableau.

CSGen numérote les zones, de manière interne, en commençant au coin supérieur gauche du corps de l'écran et incrémente le numéro de la zone de gauche à droite et de haut en bas. A la figure 3.1. se trouvent deux exemples illustrant cette numérotation; le premier représente un écran de saisie "normal", le second représente un tableau

Les touches de fonction réservées au cheminement vers la zone suivante sont TAB et "flèche vers le bas" alors que vers la zone précédente, les touches utilisées sont Shift-TAB et "flèche vers le haut". Dans le second exemple de la figure 3.1., il est plus logique que la touche "flèche vers le haut" conduise à la zone de saisie située au dessus de la zone courante plutôt qu'a la zone située à gauche de la zone courante (celle-ci est sa précédente numérique). Donc, selon la disposition géométrique de l'écran, ces touches ont un fonctionnement différent, ou encore la notion de succession est modifiée.

Dans l'exemple du tableau, il faut utiliser d'autres touches pour accéder à la zone située à gauche ou à droite de la zone courante. Il s'agit respectivement de F5 ou "Ctrl-flèche gauche" et F6 ou "Ctrl-flèche droite". En effet, les touches "flèche gauche" et "flèche droite" ont une autre fonction comme nous le verrons ci-dessous.
Une zone de saisie est validée par les touches "Return" ou "Enter" provoquant le passage à la zone de saisie suivante.

Les touches de fonction "Ctrl-PgUp" et "Ctrl-PgDn" permettent respectivement d'accéder à la première et à la dernière zone de l'écran affiché.

Les touches utilisées pour l'édition au sein d'une zone de saisie sont définies ci-dessous :

Flèche gauche	Déplacement du curseur vers la gauche.
Flèche droite	Déplacement du curseur vers la droite.
Home	Positionnement du curseur sur le premier caractère de la zone.
End	Positionnement du curseur sur le dernier caractère de la zone.
Back space	Effacement du caractère à gauche du curseur.
Del	Effacement du caractère sous le curseur.
Ins	Insertion d'un caractère blanc sous le curseur.
Ctrl-Home	Mise à blanc de la zone de saisie.

GURE 3.1. Exemples de numérotation de zones

CREATION CLIENT : SAISIE SIGNALETIQUE

Nom du client ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒ (0)

Prénom ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒ (1)

Adresse ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒ (2)

PTT-Localité ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒ (3)

Naissance ▒▒-▒▒-▒▒ (4)

F10 = Acceptation et création Esc = Retour menu

CREATION CLIENT : SAISIE FAMILLE

Nom du client ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒ (0)

Prénom enfant		Date naiss.		Commentaire	
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒	(1)	▒▒-▒▒-▒▒	(2)	▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒	(3)
	(4)	▒▒-▒▒-▒▒	(5)		(6)
	(7)	▒▒-▒▒-▒▒	(8)		(9)
	(10)	▒▒-▒▒-▒▒	(11)		(12)
	(13)	▒▒-▒▒-▒▒	(14)		(13)

F10 = Acceptation et création Esc = Retour menu

3.15 Les primitives de CSGen

Après cette approche de la programmation nous allons énumérer les primitives que fournit CSGen pour manipuler les écrans, en les commentant brièvement.

Pour respecter la philosophie du langage C, chaque fonction retourne une valeur. Les fonctions provoquant la saisie d'un écran ou l'attente d'un caractère (Fonctions marquées *) renvoient une valeur représentant la touche de sortie utilisée par l'utilisateur. Les autres retournent une valeur indiquant si la fonction s'est correctement déroulée ou non (erreur dans un fichier, ...).

Nom	Commentaire

Cnewst(nom écran)
: Initialisation de la pile d'écrans et push d'un écran

Cnewsted(nom écran)
*
: Initialisation de la pile, push d'un écran et édition

Cpsh(nom écran)
: push d'un écran au sommet de la pile.

Cpshed(nom écran)
*
: push d'un écran au sommet de la pile et édition.

Cmenu(nom menu)
*
: Initialisation de la pile, affichage d'un menu et "édition" de celui-ci.

Cpop(nom écran)
: pop de l'écran au sommet de la pile.

Cpoped(nom écran)
*
: pop de l'écran au sommet de la pile et édition.

Ctop()
: restaure les zones de saisie de l'écran au sommet de la pile.

Ctoped()
*
: restaure et édite les zones de saisie de l'écran au sommet de la pile.

Csavest()

sauvetage de la pile courante pour réutilisation ultérieure (utile lors d'un changement de contexte).

Crsted()
*

Suppression de la pile courante et restauration de la pile sauvegardée par Csavest().

Csgopen()
Csgclose()

Initialisation et clôture d'une session de travail.

Cerror(type erreur,message erreur)

Affichage d'un message d'erreur dans la zone erreur et type d'erreur.

Ceerror()

Effacement de la zone erreur et type d'erreur.

Cperror(type erreur,message erreur)
*

Combinaison de Cerror, d'une pause et de Ceerror.

3.16 Fonctions de tests de validité.

Les tests de validité sont menés lors de la sortie d'un écran par une touche de fonction différente de Esc.

Les fonctions de tests de validité qui réalisent une comparaison renvoient une valeur prédéfinie "OK" ou "ERR" qui correspondent respectivement aux valeurs "TRUE" et "FALSE" du langage C. Ces fonctions sont distinguées par leur préfixe "is". Les autres fonctions énoncées ci-dessous retournent une valeur de type "char*".

La syntaxe de ces tests est très libre puisqu'ils sont écrits en langage C. Le programmeur peut donc les utiliser à d'autres fins que la simple vérification syntaxique ou sémantique. Une application possible de ces tests est d'altérer la valeur d'une variable ou d'une zone de saisie. En effet, par défaut, CSGen renvoie une zone de saisie telle qu'elle est affichée c'est-à-dire complétée par des caractères "blanc". c'est la raison pour laquelle certaines fonctions de test de validité suppriment les blancs à gauche ou à droite d'une variable liée à une zone de saisie.

Nom	Commentaire

isempty(chaîne de caractères)
Teste si une chaîne de caractères est vide.

isblank(chaîne de caractères)
Teste si une chaîne de caractères n'est composée que de caractères "blanc".

isdigit(chaîne de caractères)
Teste si une chaîne de caractères n'est composée que de caractères numériques.

isdate(chaîne de caractères)
Teste si une chaîne de caractères est une date valide au format jj-mm-aa.

isbank(chaîne de caractères)
Teste si une chaîne de caractères est un numéro de compte bancaire valide au format nnn-nnnnnnnn-nn.

isequal(chaîne de caractères,chaîne de caractères)
Converti les chaines de caractères en nombre entier et vérifie s'ils sont égaux.

islower(chaîne de caractères,chaîne de caractères)
Converti les chaines de caractères en nombre entier et vérifie si le premier nombre est inférieur au second.

isgreater(chaîne de caractères,chaîne de caractères)
Converti les chaines de caractères en nombre entier et vérifie si la premier nombre est supérieur au second.

trim(chaîne de caractères)
Supprime les "trailing blanks" d'une chaîne de caractères.

ltrim(chaîne de caractères)
Supprime les "leading blanks" d'une chaîne de caractères.

ttrim(chaîne de caractères)
Supprime les "leading" et "trailing blanks" d'une chaîne de caractères.

upper(chaîne de caractères)
converti une chaîne en "upper case".

lower(chaîne de caractères)
converti une chaîne en "lower case".

3.17 Les programmes constituant CSGen

Nous n'allons pas entrer dans le détail de l'utilisation des programmes de CSGen, mais nous allons brièvement décrire leurs fonctions et énoncer les données échangées entre ces composants.

3.18 L'éditeur d'écrans.

Le programme GET est l'éditeur d'écran. Il permet de définir un écran de fond, des écrans de type menu, des écrans de dialogues et des écrans d'aide.

Les fonctionnalités de cet éditeur sont semblables aux fonctionnalités offertes par beaucoup d'autres éditeurs ou traitements de textes (édition "full screen", insert et delete line, déplacement du curseur, move et copy block, tracé automatique de cadres, ...).
Les différences les plus sensibles avec ses "congénères" est la limitation de l'édition à une seule page d'écran, la prise en compte des zones de saisies et des zones de fond.

Pratiquement, l'édition de l'écran de fond s'effectue en deux temps :

- Sur l'entièreté de l'écran le développeur définit l'aspect de l'écran de fond (cadres, commentaires, ...).

- Le développeur définit ensuite la taille et la position des zones de fond.

L'édition des écrans logiques s'effectue zone de fond par zone de fond (dialogue,titre,options), ensuite s'effectue la définition des touches de sortie de l'écran. Lors de l'édition de la zone de dialogue, une touche de fonction permet l'affichage d'un caractère spécial, qui est reconnu par l'éditeur comme "caractère de définition d'une zone de saisie". Après édition de l'écran logique, l'éditeur oblige le développeur a préciser les zones de saisie ainsi définies (nom de variable,type,format,test de validité,message d'erreur).

L'éditeur d'écran crée des fichiers représentant les divers types d'écran édités, les fichiers-écrans, en voici une brève description formalisée à l'aide des structures du langage C:

Fichier-écran de fond.

```
struct {
          int ligne;
          int colonne;
          char texte(80);
       }
    texte_fond();

struct {
          int ligne_début;
```

```
            int colonne_début;
            int ligne_fin;
            int colonne_fin;
            int attribut;
        }
    déf_zone_fond();
```

La première structure représente le texte contenu par l'écran de fond, la seconde contient la définition des zones de fond.

Fichier-écran logique.

```
struct {
            int ligne;
            int colonne;
            char texte(80);
        }
    texte_logique();

struct {
            int ligne;
            int colonne;
            int longueur;
            char type_zone;
            char nom_variable(30);
            char format_zone(80);
            char test_zone(80);
            char message_erreur(80);
        }
    zone_de_saisie();

int touches_de_sortie(10);
```

Le premier groupe représente le texte affiché par cet écran, le second contient les renseignements relatifs aux zones de saisie et le troisième groupe définit les touches de sortie de cet écran.

Fichier-écran de menu.

```
struct {
            int ligne;
            int colonne;
            char texte(80);
        }
    texte_menu();

struct {
            int ligne;
            int colonne;
            int longueur;
```

```
            char abbréviation_choix(5);
            int valeur_retour_choix;
            char commentaire_choix(80);
        }
    point_de_menu();
```

Le premier groupe représente le texte affiché par l'écran de menu, le second définit les points du menu.

Fichier-écran d'aide d'un écran logique.

```
struct {
            int ligne;
            int colonne;
            char texte(80);
        }
    texte_aide();
```

Il faut également noter que l'éditeur d'écrans crée un fichier contenant les noms de tous les fichiers des écrans logiques utilisés pour réaliser une application. Il est possible, de cette manière, de connaître les noms de fichiers-écrans sans utiliser les fonctions C d'accès aux directory, Celles-ci n'étant pas nécessairement portables car trop dépendantes du système d'exploitation.

3.19 Les fichiers de définition.

Pour assurer une portabilité aisée, CSGen utilise des fichiers de définition de l'environnement de travail.

- Un fichier définit le système d'exploitation employé. S'il s'agit du MS-DOS, des paramètres décrivent l'emplacement en mémoire de la carte d'affichage, de manière à court-circuiter MS-DOS dont la gestion d'écran n'est pas des plus rapides !

- Un fichier définit les "Escape sequences" liées aux fonctions d'affichage au terminal (Attributs d'affichage, Positionnement sur l'écran, Effacement de l'écran, ...).

- Un fichier définit les codes envoyés lors de l'appui sur les touches du clavier, ce qui est particulièrement important pour l'utilisation des touches de fonction. Les touches correspondant à de simples caractères sont supposées être définies selon le code ASCII.

3.20 Le compilateur d'écrans et la "run-time library".

Le compilateur d'écran a pour but de rendre les fichiers-écrans créés par l'éditeur utilisables par un programme d'application, via la librairie de CSGen. En effet, certains paramètres définis dans les fichiers-écrans sont inutilisables par le compilateur C. Par exemple, on ne peut accéder à une variable C en utilisant directement son nom, considéré comme une constante. De la même manière, on ne peut exécuter un test de validité sans compiler son expression.
Le langage C ne fournit pas de fonction telle que DO ou EXEC("command string").
Le problème revient à extraire des fichiers-écrans les variables et les tests de validité et à les présenter au compilateur sous une forme syntaxiquement correcte.

Pour mener cette tâche à bien, nous avons joint, à chaque variable liée à une zone de saisie un numéro qui l'identifie. Il suffit alors de créer une table d'adresses indicée par le numéro d'identification; cette table donnant, pour chaque entrée, l'adresse d'une variable. On peut de cette manière accéder à une variable par son numéro. Cette table doit être créée lors de l'exécution du programme puisque le langage C effectue une allocation dynamique de la mémoire (On ne connaît l'adresse d'une variable que lors de l'exécution du programme).
Comme je l'ai déjà souligné, les variables correspondant aux zones de saisies sont globales, donc accessibles à partir de n'importe quelle fonction C.

Pratiquement :

Nous allons utiliser les écrans du second exemple du paragraphe 2.9. En tenant compte des variables définies dans les écrans, le compilateur d'écrans génèrera les fichiers sources C illustrés ci-après.

Le premier fichier définit les variables rencontrées dans les différents écrans de l'exemple.

```
struct {
         char nom[31];
         char prenom[21];
         char adresse[37];
         char ptt[5];
         char localite[31];
         char naiss[9];
         char banque[15];
       } cli ;

struct {
         char num[16];
         char design[41];
```

```
        char omschr[41];
        char achat[9];
        char vente[9];
        char tva[3];
        char qstock[5];
        char qfourn[5];
        char qclient[5];
      } art ;

char tva[7][3];

char choix[2];
```

Le second fichier définit certaines variables internes à la librairie de CSGen. Ces variables dépendent du nombre de variables définies dans les écrans de CSGen. Il eut été possible d'éviter cette génération par l'utilisation de la fonction "malloc()" (pour "memory allocation") mais certaines irrégularités de fonctionnement de cette fonction dans certains compilateurs nous ont forcé à éviter son emploi.
La variable "vadd" représente l'index vers les variables utilisées par les écrans. La fonction "Cparinit()" est appelée avant la première référence à un écran et a pour but d'initialiser l'index.

```
char *vadd[24];
extern char err_mess[];

Cparinit()
{
vadd[0]=choix;
vadd[1]=cli.nom;
vadd[2]=cli.prenom;
vadd[3]=cli.adresse;
vadd[4]=cli.ptt;
vadd[5]=cli.localite;
vadd[6]=cli.naiss;
vadd[7]=cli.banque;
vadd[8]=art.num;
vadd[9]=art.design;
vadd[10]=art.omschr;
vadd[11]=art.achat;
vadd[12]=art.vente;
vadd[13]=art.tva;
vadd[14]=art.qstock;
vadd[15]=art.qfourn;
vadd[16]=art.qclient;
vadd[17]=&tva[0][0];
vadd[18]=&tva[1][0];
vadd[19]=&tva[2][0];
vadd[20]=&tva[3][0];
vadd[21]=&tva[4][0];
```

```
vadd[22]=&tva[5][0];
vadd[23]=&tva[6][0];

csginit("M3");
}
```

Le développeur n'aura qu'à inclure cette source dans son programme et à programmer un appel à la fonction "Cparinit()" pour que CSGen puisse retrouver les variables représentées par les zones de saisie.
L'appel à la fonction "Csginit" réalise l'initialisation interne de CSGen, c'est-à-dire :

- Le chargement des fichiers de définition de l'environnement décrits au paragraphe 3.19.

- Le chargement des paramètres liés à l'application (Définition des attributs d'affichage des zones, Noms des écrans définis, ..).

- Affichage de l'écran de fond sur le terminal.

Le problème des tests peut également être résolu par le même principe. En utilisant toujours le second exemple, CSGen génère le code-source suivant :

```
test_zone(numzone)
int numzone;
{
 err_mess[0]=0;

 switch(numzone)
  {

/******************* Ecran : menu.SM3 **********************/
case 0 :
   if(choix[0]!='1'&&choix[0]!='2')
    strcpy(err_mess,"Entrez soit 1, soit 2 comme choix.");
   break;

/******************* Ecran : inkeyc.SM3 **********************/
case 1 :
   if(isempty(upper(ttrim(cli.nom))))
    strcpy(err_mess,"Le nom du client ne peut être vide.");
   break;

/******************* Ecran : inkeya.SM3 **********************/
case 8 :
   if(isempty(upper(ttrim(art.num))))
    strcpy(err_mess,"Le numéro de l'article ne peut être vide.");
   break;
```

```
/****************** Ecran : modcli.SM3 **********************/
case 26 :
   if(!inrange(cli.ptt,1000,9999))
    strcpy(err_mess,"Le code postal doit être compris entre 1000
et 9999");
   break;
case 28 :
   if(!isdate(cli.naiss))
    strcpy(err_mess,"La date de naissance est invalide (JJ-MM-
AA).");
   break;
case 29 :
   if(!isbank(cli.banque))
    strcpy(err_mess,"Le numéro de compte en banque est
invalide.");
   break;

/****************** Ecran : newcli.SM3 **********************/
case 30 :
   if(isempty(ttrim(upper(cli.nom))))
    strcpy(err_mess,"Le nom du client ne peut être vide.");
   break;
case 33 :
   if(!inrange(cli.ptt,1000,9999))
    strcpy(err_mess,"Le code postal doit être compris entre 1000
et 9999");
   break;
case 35 :
   if(!isdate(cli.naiss))
    strcpy(err_mess,"La date de naissance est invalide (JJ-MM-
AA).");
   break;
case 36 :
   if(!isbank(cli.banque))
    strcpy(err_mess,"Le numéro de compte en banque est
invalide.");
   break;

/****************** Ecran : modart.SM3 **********************/
case 40 :
   if(islower(art.vente,art.achat))
    strcpy(err_mess,"Le prix de vente ne peut être inférieur au
prix d'achat.");
   break;
case 41 :
   if(!istva(art.tva))
    strcpy(err_mess,"Le pourcentage de TVA est invalide.");
   break;

/****************** Ecran : newart.SM3 **********************/
case 49 :
   if(islower(art.vente,art.achat))
    strcpy(err_mess,"Le prix de vente ne peut être inférieur au
prix d'achat.");
```

```
   break;
case 50 :
   if(!istva(art.tva))
    strcpy(err_mess,"Le pourcentage de TVA est invalide.");
   break;

  }

 if(err_mess[0] == 0) return(1);
                 else return(0);
}
```

On retrouve dans ce listing chaque test de validité défini dans les écrans de l'exemple. Chaque test est précédé de la primitive "case" dont l'argument est fourni par une variable interne à CSGen, cette variable constitue un index vers les tests et est créée par le compilateur d'écran décrit au paragraphe 3.20.
Il suffira à CSGen d'appeler la fonction "test" pour déterminer si une zone de saisie a été correctement éditée par l'utilisateur.

Pour être complet, il reste à lier la numérotation établie et les fichiers-écrans. Pour ce faire, le compilateur d'écrans crée un ensemble de fichiers qui reprennent la structure des fichiers-écrans de dialogue, et qui seront utilisés par la librairie de CSGen (on peut les appeler fichiers-écrans compilés). En voici la forme exprimée sous la forme de structures en langage C:

```
struct {
          int ligne;
          int colonne;
          char texte(80);
       }
     texte_écran();

struct {
          int ligne_début;
          int colonne_début;
          char format_saisie();
          int numéro_de_variable;
       }
     déf_zone_saisie_compilee();

struct {
          int code_touche_de_fonction;
       }
     touche_de_sortie();
```

La première structure représente le texte contenu par l'écran, la seconde contient la définition des zones de saisie et leur association aux variables C.

Il suffit donc de remplacer dans les fichiers-écrans les noms de variables par le numéro identificateur créé. Ce numéro peut alors être utilisé pour accéder, via la table des adresses, à la variable correspondant à la zone de saisie. Par exemple, "choix" sera remplacé par 0, "cli.nom" par 1,...
On remarque que le fichier-écran compilé ne reprend pas les données déjà exploitées dans les sources générées.

3.21 Le programme d'impression.

Le programme d'impression permet d'obtenir la copie sur papier de toutes les images-écrans d'une application, accompagnées de l'impression des attributs des zones de saisie qu'elles contiennent. Ce programme imprime également une "cross reference" des écrans et des zones de saisie qui y sont définies. Un message d'erreur ("Warning") est imprimé lorsqu'une même variable possède des longueurs différentes dans différents écrans. Il est enfin possible d'envoyer ces impressions dans un fichier, de manière à les utiliser dans un traitement de texte puisqu'il s'agit d'une documentation accompagnant le dossier d'analyse de l'application.

3.22 Exemple de programme utilisant CSGen utilitaire.

Voici, ci-après, un listing montrant comment les primitives de CSGen utilitaire peuvent être employées pour programmer le second exemple.

On y retrouve les fichiers générés par le compilateur d'écrans, sous les noms "Csgenm3.h" et "Csgenm3.hst". Le fichier "Benkeys.h" définit les codes des touches de fonction tels que TI_F1, TI_F2, TI_CAN, ...
On suppose dans ce programme avoir un utilitaire de gestion de fichier dont les fonctions s'appellent "Dopen", "Dread", "Dexist", .. Ces fonctions sont données à titre d'exemple et ne sont pas explicitées, leur syntaxe étant triviale.

```
#include ‹stdio.h›
#include ‹csgenM3.hst›
#include ‹csgenM3.h›

#include ‹benkeys.h›

#define CLI 0
#define ART 1

main()
```

```
{
 int st;

 csgopen();

 do
  {
   st = Cnewsted("MENU");

   if(st != TI_CAN)
    {
     switch(choix[0])
      {
       case '1' : ges_client(CLI);
                  break;

       case '2' : ges_article(ART);
                  break;
      }
    }
  }
 while(st != TI_CAN);

 csgclose();
}


ges_client()
{
 int key,keys;

 Dopen(CLI);

 key = Cnewsted("INKEYC");

 while(key != TI_CAN)
  {
   switch(key)
    {
     case TI_FA : if(Dexist(CLI,cli.nom)) Dread(CLI,cli.nom);
                     else Cperror("DBCLI","Le client n'existe
pas");

                  break;

     case TI_F2 : if(! Dexist(CLI,cli.nom))
                   {
                    keys = Cpshed("NEWCLI");

                    if(keys != TI_CAN) Dadd(CLI);

                    Cpop();
                   }
                  else Cperror("DBCLI","Le client existe déjà");
```

```
                    break;

     case TI_F3 : if(Dexist(CLI,cli.nom))
                   {
                    keys = Cpshed("MODCLI");

                    if(keys != TI_CAN) Drew(CLI);

                    Cpop();
                   }
                  else Cperror("DBCLI","Le client n'existe pas");

                  break;

     case TI_PGD : if(Dnext(CLI,cli.nom) == NOKEY)
                      Cperror("DBCLI","Fin du fichier atteinte");

                   break;

     case TI_PGU : if(Dprev(CLI,cli.nom) == NOKEY)
                      Cperror("DBCLI","Début du fichier
atteinte");

                   break;

     case TI_F7 :  if(Dexist(CLI,cli.nom)) Dsupp(CLI,cli.nom);
                      else Cperror("DBCLI","Le client n'existe
pas");

                   break;
    }
   key = Ctoped();
  }
 Dclose(CLI);
}


ges_article()
{
 int key,keys;

 Dopen(CLI);

 key = Cnewsted("INKEYA");

 while(key != TI_CAN)
  {
   switch(key)
    {
     case TI_FA : if(Dexist(ART,art.num)) Dread(ART,art.num);
                     else Cperror("DBART","L'article n'existe
pas");
```

```
                    break;

      case TI_F2 : if(! Dexist(ART,art.num))
                    {
                     keys = Cpshed("NEWART");

                     if(keys != TI_CAN) Dadd(ART);

                     Cpop();
                    }
                   else Cperror("DBART","L'article existe déjà");

                   break;
      case TI_F3 : if(Dexist(ART,art.num))
                    {
                     keys = Cpshed("MODART");

                     if(keys != TI_CAN) Drew(ART);

                     Cpop();
                    }
                   else Cperror("DBART","L'article n'existe pas");

                   break;

      case TI_PGD : if(Dnext(ART,art.num) == NOKEY)
                       Cperror("DBART","Fin du fichier atteinte");

                    break;

      case TI_PGU : if(Dprev(ART,art.num) == NOKEY)
                       Cperror("DBART","Début du fichier
atteinte");

                    break;

      case TI_F7 :  if(Dexist(ART,art.num)) Dsupp(ART,art.num);
                     else Cperror("DBART","L'article n'existe
pas");

                    break;
     }
    key = Ctoped();
   }
  Dclose(CLI);
}
```

CHAPITRE 4 : CSGEN SUPERVISEUR.

La version utilitaire de CSGen est utilisée au sein de la société Prologic depuis environ six mois. Elle a permis de développer différentes applications telles que diverses gestions de fichier, une gestion de cabinet médical, une comptabilité forfaitaire et une comptabilité générale à l'état encore embryonnaire.
Cette confrontation avec les besoins réels a permis de mieux cerner les avantages et les inconvénients du gestionnaire d'écrans.

Outre les inconvénients, des améliorations ont été suggérées par les programmeurs utilisant CSGen, ce qui a amené à étudier une nouvelle version du gestionnaire d'écrans.

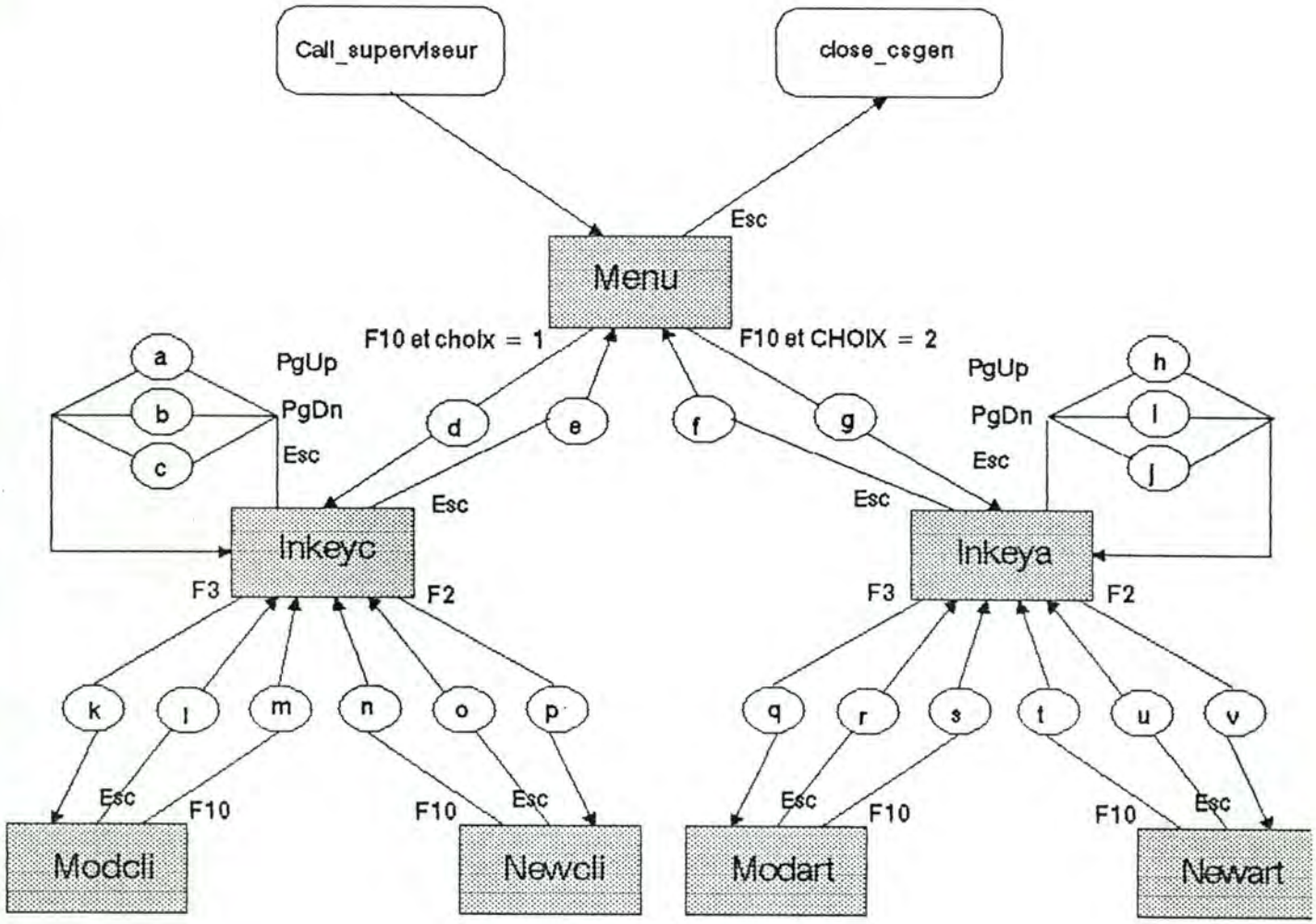
La lecture du livre de Bernard Faulle a permis de mettre en exergue des notions qui n'existent pas (ou sont déformées) dans l'approche utilitaire du gestionnaire d'écrans.
Cette lecture a également montré les nombreux avantages qu'il y avait à considérer le gestionnaire d'écran non pas comme un utilitaire mais comme le "superviseur" du programme c'est-à-dire à considérer le dialogue homme-machine non pas comme un auxiliaire mais comme l'élément pilote d'une application.

4.1 L'idée de scénario

L'approche superviseur est radicalement différente de l'approche utilitaire par la dynamique qu'elle engendre, à l'aide par exemple de la notion de scénario.
Dans ce cas, un écran logique n'est plus vu comme une unité de dialogue indépendante mais au contraire, il appartient à une suite ordonnée préétablie d'unités dialogues.
Le concept de scénario a été défini au paragraphe 2.13. La figure 4.1. illustre le scénario du second exemple tel qu'il a déjà été présenté à cet endroit.

Comme nous l'avons déjà mentionné dans le chapitre 2, les rectangles représentent des écrans logiques alors que les arcs représentent des traitements. Un gestionnaire d'écran exploitant un tel schéma peut être à même de gérer tout un programme à condition de "connaître" et de pouvoir appeler les traitements liés aux arcs, les fonctions-arcs (définies au paragraphe 2.13.).

FIGURE 4.1. Scénario de la seconde solution.



x = *Fonction – arc*

= *Ecran*

PgUp, PgDn, Esc, F2, F3, Esc = Touches de fonction

4.2 Les limitations de CSGen utilitaire

Voici les inconvénients qui ont été constatés lors de l'utilisation de la première version de CSGen. Certaines critiques constituent de simples détails mais permettront de définir une fois pour toutes les fonctionnalités attendues pour la nouvelle version du gestionnaire d'écrans.

- Difficulté de découper les images-écrans en écrans logiques.
- A l'utilisation, on s'est aperçu qu'il était parfois ardu de définir si c'était une instruction CPUSH, CPOP ou CNEWST qu'il fallait employer. Le schéma précédent légèrement complèté permet au gestionnaire d'écran de savoir quelle instruction employer, ce qui permet une disparition des appels explicites aux fonctions de manipulation d'écran.

- Une limitation est l'impossibilité de superposer proprement plusieurs écrans logiques ayant des régions de l'écran communes.

- La forte distinction entre un écran de dialogue et un écran de menu interdisant leur superposition limite beaucoup l'utilisation de menus "locaux" (par exemple "choisissez le périphérique d'impression,...). De plus le principe consistant à faire se déplacer un repère en face des choix possibles n'est pas toujours le moyen le plus rapide pour choisir un point de menu.

- CSGen utilitaire n'admet pas de valeurs de zones de saisie par défaut.

- Il est impossible de forcer le cadrage d'une zone de saisie à gauche ou à droite.

- Le format de la date n'est pas imposé mais sa validation bien car la fonction isdate() assume que la date est entrée sous le format "jj-mm-aa".

- La taille maximum de l'écran du terminal admise par CSGen est de 25 lignes de 80 caractères. Or les nouvelles cartes sous MS-DOS, ou les terminaux graphiques admettent des tailles plus importantes.

- Les écrans d'aide de CSGen apportent plusieurs limitations :
  - Chaque écran logique doit avoir au plus un écran d'aide.
  - Il est impossible d'enchaîner des écrans d'aide.
  - Un écran d'aide est une "constante de texte".

- Le nombre et la désignation des zones de fond est fixe, or certaines de ces zones (clé accédée) sont généralement inutilisées.

- Les touches de fonction propre à CSGen (zone entrée-sortie suivante, précédente, aide, DO, UNDO, ...) sont fixées ce qui pourrait limiter la portabilité ou l'ergonomie (F10 n'est pas nécessairement la touche la plus accessible !).

- Les attributs d'affichage des zones de saisie sont définis pour l'ensemble des écrans d'une application or il peut être intéressant de les modifier localement.

- La modification de la taille d'une variable ou de son format entraîne en général un important travail de modification au sein des écrans logiques utilisant cette variable.

- La création d'un écran logique se fait "en aveugle" puisque la notion d'écran physique n'existe pas de manière explicite.

- Les textes et les zones de saisie sont définis en coordonnées ligne, colonne relative au coin supérieur gauche de l'écran du terminal. Le déplacement d'une zone de fond entraîne donc des résultats inattendus comme par exemple la présence d'une zone de saisie hors de la zone de dialogue.

- La présence permanente de la zone des options est gênante car elle limite la surface utilisable de l'écran du terminal. De plus un utilisateur habitué à utiliser une application ne doit pas avoir constamment sous les yeux les options disponibles.

4.3 L'écran de fond

Les critiques de CSGen utilitaire mettent en évidence la nécessité pour le programmeur de pouvoir définir ses propres zones de fond selon l'application à réaliser.
Néanmoins, il existe des zones nécessaires au bon déroulement de CSGen (par exemple la zone des messages d'erreur) et au déroulement du dialogue (par exemple la zone corps d'écran).
Lors de la création d'un écran de fond, il faudra donc définir des zones de fond et spécifier leur rôle. Les zones de fond obligatoires sont au nombre de trois, à savoir :

- La zone corps d'écran.
- La zone des messages d'erreur (et la zone type d'erreur).
- La zone des options.

Il faut aussi distinguer les zones de fond "éditables" par le programmeur de celles qui ne le sont pas. Par exemple, la zone corps d'écran ainsi qu'une éventuelle zone titre sont éditables, alors que la zone des messages d'erreur ne l'est pas.

Pour être tout à fait complet, il faut déterminer les zones qui sont entièrement effacées à chaque affichage d'écran logique. Le problème, posé à l'envers, revient à voir quelles zones doivent être superposables. Après analyse, il ressort que seule la zone corps d'écran doit l'être, entre autres raisons de façon à ne pas

compliquer la confection des écrans. Pour cette même raison, on confinera toujours les zones de saisie à la zone corps d'écran.

Par contre, il peut être intéressant de définir des "zones de sortie permanentes" liées à l'écran de fond. Elles peuvent uniquement apparaître à un endroit de l'écran où n'y a aucune zone de fond. On pourra par exemple les utiliser pour afficher l'heure, la date du jour,... Ce moyen permet également de supprimer la zone de fond "nom d'écran", en assumant que CSGen initialise une variable globale contenant le nom de l'écran logique au sommet de la pile.

4.4 La zone des options

Les critiques de CSGen utilitaire indiquent les inconvénients à obtenir l'affichage permanent de la zone des options. Cette zone peut occuper une surface importante de l'écran, celle-ci étant alors inutilisable pour la zone de corps d'écran. Cependant la zone des options est indispensable à l'utilisation du logiciel puisqu'elle traduit le scénario de l'application. La solution retenue dans CSgen superviseur est de considérer les options comme une aide, dont l'affichage est consécutif à l'appui sur une touche de fonction.

Il faut aussi souligner l'importance des touches de sortie d'écran dans la confection de la zone des options puisqu'en effet elles sont sa seule raison d'être. Pour cette raison, CSGen superviseur propose un utilitaire en simplifiant considérablement la confection. Cet utilitaire propose au programmeur d'une part une zone des options vierge, dans laquelle celui-ci y définit les emplacements qui recevront les "noms" des touches de fonction ainsi que leur signification et d'autre part un tableau permettant d'affecter des significations par défaut aux touches de fonctions (L'ergonomie n'en sera que meilleure). A l'aide du scénario et des informations ci-avant, CSGen peut ainsi générer semi-automatiquement le texte de la zone des options de chaque écran physique.

4.5 La notion de grilles d'écran

L'utilisation de CSGen utilitaire a montré que, dans une application de gestion, beaucoup d'écrans avaient trait à de même textes ou zones de saisie positionnés au même endroit. Par exemple, les écrans logiques affichés lors de traitements de création d'un client, de modification d'un client ou encore de consultation d'un client ont la même apparence et utilisent les mêmes zones de saisie. Seuls les types (input,modif ou output) ou les tests de validité sont différents d'un écran logique à l'autre.
La création de ces différents écrans logiques devient alors une tâche répétitive et constitue dès lors une source d'erreurs.

Face à ce nouveau problème, nous avons eu l'idée non plus de diviser un écran logique en un ensemble de textes et de zone de saisie, mais de le diviser en grilles d'écrans constituées de textes et de zones de saisie.

Les éléments constitutifs de ces grilles d'écrans sont donc un sous-ensemble des éléments constituants les écrans logiques, à savoir :

- du texte situé dans la zone corps d'écran.
- des zones de saisie dans la zone corps d'écran.

  Pour chacune de ces zones :

  - Le nom de la variable C associée.
  - Le format et la longueur de la zone.

Après l'introduction de la notion de grilles d'écrans, un écran logique devient une "somme" de grilles d'écrans, accompagnées de précisions relatives à l'utilisation des zones de saisie et de textes dans les zones éditables autres que la zone corps d'écran.

4.6 Les attributs des zones de saisie

Lors de la confection des applications à l'aide de CSGen utilitaire, nous avons remarqué des dépendances fonctionnelles entre les zones de saisie et leurs attributs. Il est possible d'exploiter ces dépendances pour alléger la création des écrans d'une application.
Nous allons décrire ci-après ces dépendances fonctionnelles :

- Le format d'une zone de saisie est lié à la variable de la zone.
  Il est en effet peu probable qu'une variable soit affichée dans différents formats dans différents écrans. De plus, les caractères d'insertion permettent d'employer un même format en entrée ou en sortie sans alourdir la tâche de l'utilisateur.

- Le test de validité d'une zone de saisie et son message d'erreur sont liés à la variable et au type de la zone de saisie (In, Out, Mod).
  Comme précédemment, il est logique que dans un programme l'édition d'une variable utilise les mêmes critères dans des écrans différents.

En conclusion, lorsque le développeur utilisera dans un écran une variable déjà définie dans un autre écran, il sera possible de reprendre le format, les tests et les messages d'erreur déjà définis, de manière à accélérer l'édition des écrans.

Dans CSGen utilitaire les tests étaient utilisés pour appliquer des fonctions de formatage à postériori de la zone de saisie. Une telle approche constitue un amalgame de deux actions fonctionnellement différentes. Il en résultait d'ailleurs des

tests lourds, par exemple, le test de validité d'une zone de saisie dans laquelle l'utilisateur doit entrer les lettres 'O' ou 'N' se traduisait par :

```
Test : "strcmp(upper(=),"O")!=0 && strcmp(upper(=),"N")!=0"
Message d'erreur : "Entrez O ou N dans cette zone."
```

Cet exemple illustre le manque de cohésion des tests de validité par la redondance de l'appel à la fonction "upper". Nous avons donc décidé de séparer les tests de validité et les actions de formatage, ce qui a donné les fonctions à postériori. L'exemple précédent devient alors :

```
Fonction à postériori : "upper(=)"
Test : "strcmp(=,"O") != 0 && strcmp(=,"N") != 0"
Message d'erreur : "Entrez O ou N dans cette zone."
```

Enfin, nous avons trouvé intéressant de pouvoir utiliser la variable d'une zone de saisie avant son édition par l'utilisateur. Nous avons introduit les fonctions à priori qui permettent, par exemple d'attribuer une valeur par défaut à une zone de saisie. Il est également possible, par ce biais, d'entrer la valeur d'une zone de saisie à l'aide de moyens qui ne sont pas offerts par CSGen. Par exemple, le choix d'une valeur dans une table de valeurs possibles; dans ce cas, le zone de saisie possède un type "Out" une tentative d'édition de cette zone se soldera par l'appel de la fonction à priori.
Il faut remarquer que la "fonction à priori" est le seul attribut d'une zone de saisie de type "Out" puisqu'une telle zone n'est pas réellement éditable. Les éventuels tests de validité seront menés par la fonction à priori.

En résumé, nous allons énoncer les attributs d'une zone de saisie :

```
Nom de variable.
Taille : Nombre de ligne, nombre de colonnes.
Format.
Type In : Fonction à priori, fonction à postériori.
          Tests de validité, messages d'erreur associés.
Type Mod: Fonction à priori, fonction à postériori.
          Tests de validité, messages d'erreur associés.
Type Out: Fonction à priori.
```

4.7 La zone d'effacement

Les critiques de CSGen utilitaire et l'utilisation d'un scénario permettent de supprimer l'utilisation de primitives d'accès aux écrans logiques. Il faut néanmoins que les routines de CSGen superviseur puissent éliminer toute ambiguïté sur la méthode à employer pour afficher un écran. Ce problème se pose tout particulièrement pour effectuer la distinction entre les rôles

joués par les primitives CPUSH et CNEWST, ainsi que pour résoudre les problèmes de superposition d'écrans logiques.

Notre idée est de supprimer la nécessité d'employer la primitive CNEWST en créant une pile vide lors du lancement de l'application et en empilant ensuite tous les écrans logiques.
En agissant de la sorte, un problème de superposition se pose lorsque l'on désire effacer totalement le contenu de la zone "corps d' écran" pour changer de contexte.
Une solution possible est alors d'associer à chaque écran logique une "zone d'effacement". Cette zone détermine ainsi la portion de corps d'écran qui sera effacée avant l'affichage de l'écran logique, permettant également de mieux définir un éventuel texte ou zone de saisie à sauvegarder.
Néanmoins, cette solution souffre de critiques liées entre autres aux efforts supplémentaires nécessaires au programmeur pour imaginer les enchaînements d'écrans. En effet, la démarche du programmeur doit être : "Si j'utilise tel écran sachant que c'est tel autre écran qui est affiché, il ne me reste plus qu'a faire afficher tel et tel texte puisque tel autre texte est déjà affiché !"
Le paragraphe suivant apporte une solution à ce problème.

4.8 Uniquement des écrans physiques

Une des critiques de la version utilitaire de CSGen est de devoir créer un écran logique "en aveugle" sans pouvoir visualiser exactement l'écran tel qu'il se présentera. Les dessins des écrans logiques du second exemple illustrent cette constatation.
Dans CSGen utilitaire, il était impossible d'agir autrement puisqu'un écran était une entité indépendante. Par contre, le scénario permet de cheminer à travers les écrans logiques et, en conjonction avec les zones d'effacement, il est possible de reproduire lors de la création d'un écran, la vue qu'aura l'utilisateur final lors de l'utilisation du programme.

Il existe une autre solution possible qui serait radicalement différente des autres : Ne pas éditer d' écrans logiques mais éditer des écrans physiques. Dans ce cas, les écrans logiques deviennent un objet interne de CSGen dont le seul but est d'augmenter les performances du logiciel. Il en résulte que la notion de grilles d'écrans disparaît, ou plus précisément, se ramène à la notion d'écrans logiques précédemment décrites.
Les problèmes de superpositions sont également résolus puisque CSGen , en connaissant l'écran physique affiché et l'écran physique à afficher, détermine lui-même les portions d'écran à modifier sur le terminal et ceci avec une efficacité maximum. On peut parler, lors de l'exécution de l'application, d'affichage différentiel.
L'utilisation de grilles d'écrans permet de confectionner directement des écrans physiques sans exiger plus de travail de la part du programmeur. En effet, la création d'un écran physique se limite à

- L'association d'une ou plusieurs grilles d'écran
- La précision du type des zones de saisie
- L'édition des zones éditables ("titre", "options",..).

Cette méthode apporte un autre avantage , par exemple lors de la modification d'un signalétique. Il suffit de modifier une grille d'écran pour qu'automatiquement soient modifiés tous les écrans physiques utilisant le nouveau signalétique. Cet avantage est important car il apporte un énorme gain de temps par rapport à CSGen utilitaire qui obligait le programmeur à rééditer chaque écran logique.

4.9 Les fonctions-arc, le passage d'arguments

Dans la version utilitaire de CSGen les variables associées à une zone de saisie sont toutes globales. Cette approche n'est pas la plus efficace mais supprime tous les problèmes de passage d'arguments aux primitives d'accès aux écrans. Il doit en être de même pour la nouvelle version de CSGen et pour les même raisons.
Pour comprendre le problème, il faut savoir qu'il n'existe en langage C que trois types de variables :

- Des variables globales au programme.
- Des variables locales à une fonction.
- Des variables locales à un ensemble d'instructions inscrites dans une fonction.

La durée de vie d'une variable locale est limitée à l'exécution de la fonction qui l'emploie et n'est accessible qu'a l'intérieur de cette fonction (sauf bien sûr si elle figure en tant que paramètre dans l'appel d'une fonction).
Il se pose alors un problème pour qu'une fonction-arc puisse passer des arguments à la fonction-arc suivante. En effet, l'absence de primitive d'accès aux écrans logiques résulte en l'affichage d'un écran logique lors d'une terminaison de fonction-arc, l'appel à la fonction-arc étant en fait effectué par le "superviseur" de CSGen. Les arguments passés à une fonction-arc ne peuvent donc être que de deux types :

- Des variables locales au superviseur.
- Des variables globales.

Il faut noter que ces deux types de variables ont sensiblement la même efficacité puisque l'exécution du superviseur commence et se termine pratiquement en même temps que l'exécution du programme.
De plus, il est impensable d'écrire ou de générer des définitions de variables locales au superviseur puisque celles-ci ne pourraient être exploitées qu'en le compilant, impliquant une perte de temps (la compilation) et un manque de protection (en devant fournir les sources du générateur).

Une problématique importante est liée aux fonctions-arcs : le morcellement du programme en une kyrielle de petites fonctions.

Pour illustrer ce fait, nous allons reprendre l'exemple de la gestion des fichiers clients et articles, en le programmant selon les règles définies ci-dessus.

```
menu_cli()
{
if(Dopen(CLI) != OK)
 {
  display_error("Erreur d'ouverture du fichier clients");
 }
}

menu_art()
{
if(Dopen(ART) != OK)
 {
  display_error("Erreur d'ouverture du fichier articles");
 }
}

cli_menu()
{
 Dclose(CLI);
}

art_menu()
{
 Dclose(ART);
}

cli_mod()
{
 readcli();
}

mod_cli_ok()
{
 Drew(CLI);
}

mod_cli_bad()
{
}

art_mod()
{
 readart();
}

mod_art_ok()
{
 Drew(ART);
}
```

```
mod_art_bad()
{
}

cli_new()
{
 initcli();
}

new_cli_ok()
{
 Dadd(CLI);
}

new_cli_bad()
{
}

art_new()
{
 initart();
}

new_art_ok()
{
 Dadd(ART);
}

new_art_bad()
{
}

read_cli()
{
 readcli();
}

read_next_cli()
{
 if(Dnext(CLI) != OK)
  {
   display_error("Impossible de lire le client suivant");
  }
}

read_prev_cli()
{
 if(Dprev(CLI) != OK)
  {
   display_error("Impossible de lire le client précédent");
  }
}
```

```
read_art()
{
 readart();
}

read_next_art()
{
 if(Dnext(ART) != OK)
  {
   display_error("Impossible de lire l'article suivant");
  }
}

read_prev_art()
{
 if(Dprev(ART) != OK)
  {
   display_error("Impossible de lire l'article précédent");
  }
}

readcli()
{
 if(Dread(CLI) != OK)
  {
   display_error("Ce client n'existe pas");
  }
}

readart()
{
 if(Dread(ART) != OK)
  {
   display_error("Cet article n'existe pas");
  }
}
```

Une nouvelle fois, on suppose, dans cet exemple, que l'on dispose d'un gestionnaire de fichier dont la syntaxe des primitives est triviale.

Avant de continuer, il est intéressant de décrire brièvement le fonctionnement du superviseur. Nous allons énoncer les routines du superviseur à l'aide du (pseudo) langage C utilisé jusqu'a présent, de manière à observer les problèmes de passages d'arguments.
Les noms des fonctions utilisées sont fictifs mais décrivent suffisamment le traitement effectué.

```
call_superviseur()
{
 clear_screen();
```

```
 load_scénario();
 load_and_display_fond();

 current_screen = first_of_scénario;

 while(current_screen != NIL)
  {
   out_key = load_display_edit(current_screen);

   current_screen = call_function_next(current_screen,out_key);
  }
}

call_function_next(screen,key)
char *screen,key;
{
 switch(screen)
  {
   case MENU :
    {
    switch(key)
     {
      case F10 : if(choix(0) == '1') menu_cli(); break;
                 if(choix(0) == '2') menu_art(); break;
      case ESC : return(NIL); break;
     }
    }
    break;

   case INKEYC :
    {
    switch(key)
     {
      case F10 : read_cli(); break;
      case ESC : end_cli(); break;
      case PGU : read_next_cli(); break;
      case PGD : read_prev_cli(); break;
      case F2  : new_cli(); break;
      case F3  : mod_cli(); break;
     }
    }
    break;

   case INKEYA :
    {
    switch(key)
     {
      case F10 : read_art(); break;
      case ESC : end_art(); break;
      case PGU : read_next_art(); break;
      case PGD : read_prev_art(); break;
      case F2  : new_art(); break;
      case F3  : mod_art(); break;
     }
```

```
    }
    break;

  case MODCLI :
    {
    switch(key)
     {
      case F10 : cli_mod_ok(); break;
      case ESC : cli_mod_bad(); break;
     }
    }
    break;

  case NEWCLI :
    {
    switch(key)
     {
      case F10 : cli_new_ok(); break;
      case ESC : cli_new_bad(); break;
     }
    }
    break;

  case MODART :
    {
    switch(key)
     {
      case F10 : art_mod_ok(); break;
      case ESC : art_mod_bad(); break;
     }
    }
    break;

  case NEWART :
    {
    switch(key)
     {
      case F10 : art_new_ok(); break;
      case ESC : art_new_bad(); break;
     }
    }
    break;

  }
}
```

On peut remarquer que la fonction "call_function_next" peut être générée par CSGen et compilée en même temps que l'application. Cette méthode permet d'appeler les fonctions-arcs à partir du langage C, sans artifice tel que des conversions d'adresses à l'exécution. En effet, cette fonction n'est formée que d'informations disponibles dans le scénario.

On peut également noter que seule la fonction "call_function_next" doit exister en source, les routines du superviseur peuvent être disponibles sous la forme d'une librairie assurant par conséquent la protection du générateur d'écran.

Il est possible de simplifier la programmation des applications utilisant CSGen superviseur en supprimant, par exemple, les définitions de fonctions qui ne possèdent pas d'instructions (mod_cli_bad(), new_art_bad(),...).

4.10 L'acceptation des écrans.

Nous avons jusqu'à présent considéré que l'acceptation ou le refus d'un écran était consécutif à la pression sur une touche de fonction. Cette vue, introduite par CSGen utilitaire permettait de mieux fixer les idées. Il faut en fait voir la notion d'acceptation d'un écran comme globale à un dialogue et non comme locale à un écran.
A cette fin, il faut étudier de plus près les conséquences de l'édition d'un écran sur la valeur des variables du programme.
Il est logique que lorsqu'un écran est refusé par l'utilisateur, les variables modifiées par celui-ci reprennent la valeur qu'elles avaient avant l'édition de cet écran. L'édition des zones de saisies d'un écran devrait avoir lieu dans un "buffer", celui-ci ne serait affecté aux variables que lors de l'acceptation de l'écran; dans le cas contraire, aucune affectation n'aurait lieu.

La méthode exposée ci-dessus impose l'utilisation d'un "buffer" sous forme d'une pile lorsque plusieurs écrans sont enchaînés, comme dans le cas du scénario représenté à la figure 3.2. Ce scénario illustre une gestion de fichier client dans laquelle un enregistrement contient le signalétique du client ainsi que les caractéristiques du matériel qu'il utilise. Dans cet exemple il se pose un problème lorsque l'utilisateur valide l'écran Modmat et, ensuite, refuse l'écran Modcli. Dans ce cas, en effet, le désir de l'utilisateur est d'enregistrer la description du matériel du client et de refuser les modifications apportées au signalétique. Nous pouvons alors constater un problème :

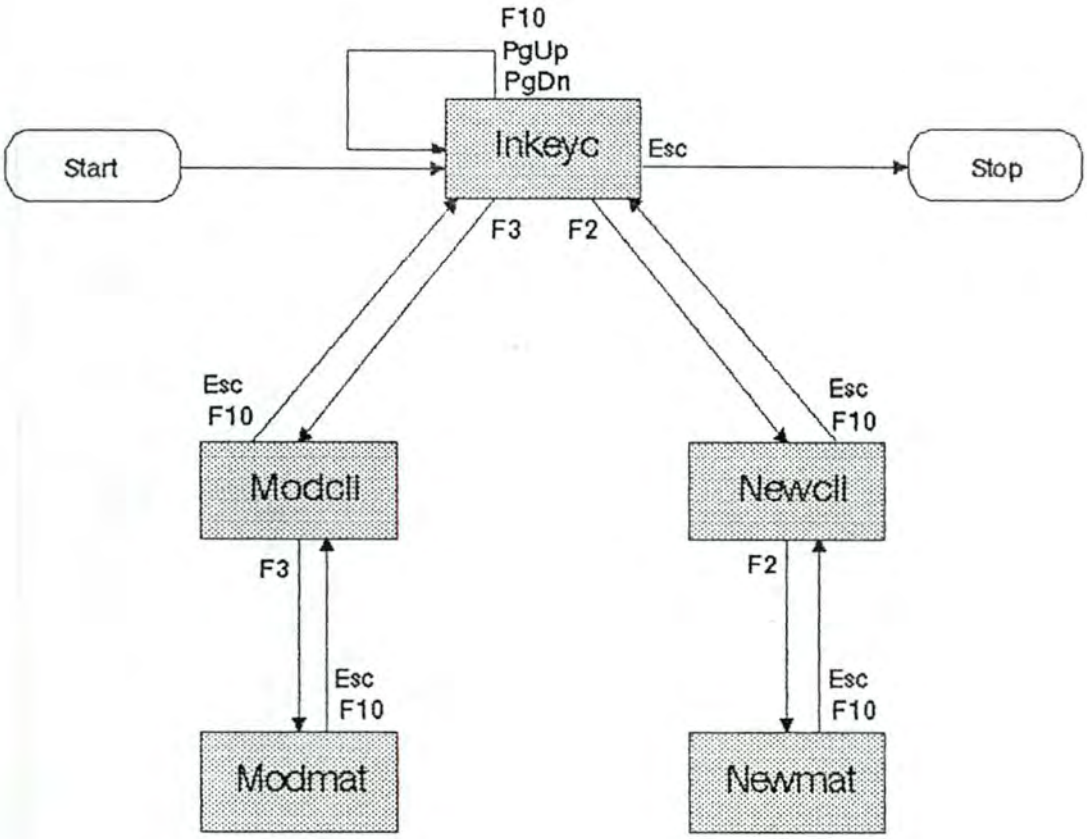
- Les informations appartenant à un même enregistrement, logiquement un seul ordre d'écriture dans le fichier doit apparaître dans le programme et forcément lors de la validation de l'écran Modcli. Cet écran étant refusé par l'utilisateur, aucune mise à jour du fichier n'aura lieu. Il y aura donc discordance entre l'enregistrement situé en mémoire centrale et l'enregistrement se trouvant dans le fichier.

Il est difficile d'échapper à ce problème, une solution étant d'accepter la discordance tout en prévenant le programmeur de manière à en éviter les conséquences. Il faut noter que, dans le cas de l'exemple, la discordance n'aura aucune influence sur le

programme à condition de programmer un ordre de lecture de l'enregistrement lors du retour à l'écran Inkeyc.

Nous avons vu dans ce paragraphe que certaines touches de fonction provoqueront la mise à jour des variables liées aux zones de saisie, alors que d'autre ne le feront pas. Il faut donc ajouter un attribut aux touches de fonction définies dans le scénario qui indiquera pour chacune d'elles le mode de mise à jour des zones de saisie de l'écran. Nous appelerons cet attribut "mise-à-jour".

FIGURE 4.2. Exemple de problème d'acceptation d'écran.



4.11 L'efficacité de CSGen superviseur.

Pour revenir au problème du morcellement du programme, on s'aperçoit d'une (quasi) disparition des variables locales, au profit de variables globales.
Il faut remarquer que la perte de place mémoire due à la "globalisation" des variables est en partie compensée par la simplification des fonctions, c'est-à-dire par un code plus compact.

En fait, le danger le plus important dû au morcellement du programme est son absence de signification s'il n'est pas accompagné de son scénario. En effet la perte ou la destruction du scénario rendra le programme tout à fait incohérent puisque c'est lui qui relie toutes les fonctions du programme.
Il est donc nécessaire pour le développeur, plus encore que pour d'autres méthodes, de bien documenter le programme. Sinon sa maintenance sera impossible. Heureusement, le générateur d'écran peut fournir automatiquement une documentation imprimée très complète du scénario, des écrans physiques et logiques ainsi que des variables employées.
Dans l'exemple donné ,on peut remarquer que les fonctions sont en général très courtes. Il est parfois plus long d'éditer leur nom que d'éditer leur corps ! On pourrait envisager de ne pas limiter le générateur à appeler une fonction-arc, mais à autoriser le développeur à éditer le corps de cette fonction. Il est fastidieux de donner un nom à une fonction alors que celle-ci peut être identifiée par l'écran dont elle est issue. Il suffit d'ajouter au programme d'édition des écrans une routine d'édition de texte.

4.12 Compatibilité avec d'autres langages évolués.

Il est certain que les choix faits au cours de l'analyse de CSGen se basent sur le langage C alors que beaucoup de gestionnaires d'écrans existants font abstraction du langage de programmation utilisé, ce qui en permet évidemment une plus large diffusion.

Pour examiner une éventuelle possibilité d'utiliser CSGen avec d'autres langages, il faut distinguer les composants qui utilisent spécifiquement le langage C.

4.13 Les fonctions du gestionnaire.

Les appels de fonctions appartenant à la librairie de CSGen sont normalement résolus par l'éditeur de lien accompagnant le système d'exploitation.
Néanmoins, le passage des arguments dépend très fort du langage employé. Par exemple, la terminaison des chaînes de caractères par un zéro binaire est une convention propre au langage C, la plupart des autres langages ne précisent pas ce point.

Le problème de passages d'arguments peut cependant être résolu si les routines du gestionnaire sont écrites en langage C puisqu'il permet l'accès à la mémoire au byte près. Il suffit donc de réaliser différentes librairies pour les différents langages visés.

4.14 Les zones de saisie d'un écran.

On doit ici distinguer les différentes composantes des zones de saisies.

La liaison programme-écran.

CSGen utilise le nom de la variable C du programme pour lier une zone de saisie d'un écran avec le programme qui l'utilise.
Comme expliqué au paragraphe 3.20, chaque variable est identifiée par une table d'adresses et l'indice correspondant.

Cette méthode est en fait valable pour tous les langages évolués qui permettent un accès direct à une zone mémoire occupée par une variable. Les langages PASCAL et BASIC possèdent généralement des fonctions "varptr" et "varlen" pour réaliser cette tâche.
Des langages tels que COBOL et FORTRAN possèdent d'autres formes d'accès tels que "equivalence" ou "redefine". Dans ce dernier cas il faudrait apporter quelques modifications à CSGen puisque le fonctionnement de ces instructions est fondamentalement différent.
Les conventions de dénomination des variables sont fortement dépendantes du langage employé, il faut donc reprogrammer les fonctions qui déterminent le type de donnée représenté par une variable.

Les tests de validité.

La génération de code source utilisée par CSGen est envisageable pour tous les langages évolués. Il faut bien entendu générer des sources syntaxiquement correctes. De toutes manières, il est toujours possible de programmer, dans le langage choisi, un interprèteur plus ou moins puissant de tests de validité.

CHAPITRE 5 : IMPLEMENTATION DE CSGEN SUPERVISEUR.

Nous n'aborderons pas dans ce chapitre l'implémentation de CSGen utilitaire. L'intérêt de cette étude est moindre puisque CSGen utilitaire reprend en grande partie les même concepts que CSGen superviseur.
Nous allons aborder l'étude de l'implémentation à l'aide de certains points résultant directement ou indirectement de l'analyse des besoins.

5.1 Portabilité.

La notion de portabilité sous entend principalement la faculté d'adaptabilité du logiciel à son environnement hardware et software et, en outre, ses performances vis-à-vis de celui-ci.

5.2 Hardware.

Le principal facteur à ranger dans cette catégorie est l'adéquation possible de CSGen aux divers types de terminaux existants. Ce problème a été résolu par l'utilisation de fichiers "driver de terminal". Ces fichiers décrivent, pour chaque terminal, les "escape sequences" des commandes des terminaux, d'une part, et les codes liés aux touches de fonction d'autre part. Ces fichiers sont redéfinissable par le programmeur pour adapter CSGen à un type de terminal qui n'est pas encore défini.
CSGen est programmé de manière à faire abstraction du terminal employé, via des primitives d'accès qui servent d'interface entre le terminal et le programme. Ces primitives sont réalisées sous la forme de fonctions écrites en langage C. Voici les principales fonctions d'accès au terminal :

- display(ligne,colonne,chaîne de caractères);
- displayc(ligne,colonne,caractère);

Ces deux fonctions affichent sur l'écran du terminal respectivement une chaîne de caractères et un caractère au coordonnées (ligne,colonne). Cet affichage se fait en accord avec les attributs de visualisation définis par les deux fonctions suivantes :

- setattr(attribut);
- resetdisp();

La première de ces fonctions permet de définir l'attribut de visualisation qui sera employé par les fonctions d'affichage. La valeur "attribut" est codée sur 16 bits décrits ci-après. La seconde fonction place l'attribut d'affichage "au neutre", selon une valeur définie dans le fichier "driver de terminal".

Description d'un code d'attribut :

Le code d'attribut est composé de 16 bits répartis en 4 groupes de 4 bits : Le premier représente le code de couleur des

caractères, le second le code de couleur sur laquelle sont affichés les caractères, enfin les deux derniers groupes codent les attributs d'affichage tels que soulignement, vidéo inverse, clignotement, haute intensité, ... Au moins deux groupes sont nécessaires pour mener cette tâche a bien, puisque les attributs sont cumulatifs (par exemple, haute intensité et soulignement).

L'interface est également composé d'une fonction d'initialisation des paramètres du terminal. Cette fonction lit le fichier "driver de terminal" et initialise ainsi les autres fonctions d'accès au terminal. Voici la syntaxe de cette fonction :

- load_ter(nom de fichier driver de terminal);

Enfin, une fonction permet de lire les caractères tapés au clavier:

- car = pgetch();

Cette fonction attend qu'une touche soit tapée au clavier du terminal, et renvoie ensuite un byte qui représente le caractère tapé :
- Code ASCII pour les touches de caractères (‹ 127)
- Code spécial pour les touches de fonction (› 127). Ce code peut être utilisé dans un programme à l'aide d'un fichier source C qui défini les codes renvoyés. L'enfoncement de la touche F1 provoque l'envoit au programme du code TI_F1, le touche Del envoie code TI_DC, ...

Au niveau hardware, les seules exigences de CSGen vis-à-vis du terminal sont :

- Qu'il puisse afficher au moins 24 lignes de 80 caractères.
- Que la position de son curseur puisse être adressable à l'aide coordonnées (ligne,colonne) ou (colonne,ligne).

5.3 Software.

La portabilité, dans ce cas, concerne le compilateur C et ses librairies de fonctions. Pour développer un logiciel portable, il faut respecter les convention dictées par "la bible du programmeur en langage C" que constitue le livre de Kernighan et Ritchie qui définit le langage C et ses instructions. On y trouve également les originalités présentées par certains compilateurs, qui constituent donc des éléments à prendre en compte lors du développement (par exemple les divergences constatées dans les valeurs retournées par certaines fonctions, le nombre de bytes des différents types de variables, l'implantation en mémoire des structures, tableaux, ...).

Cependant, deux ensembles de fonctions utilisées par CSGen échappent à cette règle de portabilité. La première dans un but

de performance, et la seconde dans un but de simplification du développement.

Les fonctions d'accès à l'écran.

Les accès à l'écran du terminal via les fonctions offertes par le langage C "standard" se sont avérées trop lentes sous MS-DOS. Il risque d'en être de même sous d'autres systèmes d'exploitation. Pour pallier cette lenteur, nous avons fait appel à des fonctions particulières à MS-DOS. En réalité, seules les fonctions "display" et "displayc" souffrent de cette incompatibilité, aussi les avons nous isolées dans un module particulier. De toute manière, les instructions particulières ne seront appelées que si un bit particulier du fichier "driver de terminal" est positionné (Bit de compatibilité MS-DOS); sinon les instructions classiques d'entrée-sortie vers la console sont employées (cprintf, putchar,...).

Les fonctions de gestion de fichiers.

La plupart des fichiers utilisés par CSGen sont de type séquentiel et sont codés en ASCII, de manière à pouvoir être facilement transférés d'un système à un autre via une simple ligne RS-232. Cependant, dans un but de simplification du développement, nous avons utilisé un gestionnaire de fichiers séquentiels indexés. Cet utilitaire n'est pas portable, il faut donc envisager d'autres solutions :

- Programmer un gestionnaire de fichier à l'aide des intructions standard du langage C.
- Utiliser un autre gestionnaire de fichiers existant sur la machine sur laquelle on désire porter CSGen.

La deuxième solution est sans doute la plus facile à réaliser car toutes les fonctions d'accès aux fichiers séquentiels indexés sont réunies dans un module de CSGen, il suffit alors de reprogrammer ce module en fonction des primitives offertes par le nouveau gestionnaire de fichiers. De plus ce module n'implémente qu'un nombre restreint de fonctions d'accès : Read, Add, Rewrite, Delete, Next et Previous record.

5.4 Performances.

Le but principal de ce paragraphe est de mettre l'accent sur le fait que, généralement, l'accès aux terminaux se fait par l'intermédiaire de lignes dont la vitesse est lente (240 ou 480 caractères par seconde). Il a donc fallu développer CSGen de manière à éviter l'envoi sur ces lignes de caractères qui ne sont pas indispensables. Ce point, qui paraît trivial, est en fait important puisque CSGen a été développé sous MS-DOS, système d'exploitation monoposte, où les transferts à l'écran sont instantanés.

5.5 L'édition des écrans.

L'analyse de CSGen superviseur a mis en évidence l'utilisation de trois types d'objets éditables :

- L'écran de fond.
- Les grilles d'écran.
- Les écrans physiques.

Ces objets sont eux-mêmes constitués de composants élémentaires que nous allons définir ci-dessous. Ces composants constituent des "types abstraits de données" qui ne sont exploitables par le programme que par l'intermédiaire de fonctions d'accès. Cette méthode à l'avantage de ne nécessiter que des modifications très localisées dans un programme, même si les composants subissent des modifications importantes. La portabilité et la maintenance du logiciel n'en seront que facilitées, au prix d'une légère perte de performance inhérente à ce type de représentation.
Les descriptions suivantes ne sont pas formelles, en effet il ne s'agit pas ici d'exposer un dossier d'analyse mais bien d'illustrer les solutions qui ont été adoptées.

5.6 Les composants élémentaires.

Les paragraphes suivants sont divisés en sous-paragraphes qui précisent les diverses facettes de l'implémentation de ces objets.
La définition, en langage naturel, précise la constitution du composant.
La représentation externe exprime la structure d'un composant telle qu'elle sera écrite sur un support magnétique.
Les fonctions d'accès décrivent les principales opérations que l'on peut effectuer sur les composants.

Il faut préciser que l'affichage d'un écran se déroule en deux phases :
- La préparation d'un écran en mémoire centrale (sans affichage).
- L'affichage de l'écran préparé sur l'écran du terminal.

Cette distinction reflète la nécessité qu'a CSGen de se représenter l'état dans lequel se trouve l'écran du terminal. Le type abstrait "dessin d'écran" cache en fait un "écran interne" qui constitue la future image qui apparaîtra sur l'écran du terminal.
Dans les paragraphes suivants le terme "écran" signifie "l'écran interne", alors que "l'écran du terminal" fera référence à la vue que peut avoir un utilisateur regardant le terminal.

5.7 Les dessins d'écran.

Définition.

Ce premier composant est constitué par des caractères qui seront envoyés dans un ordre déterminé à un emplacement déterminé de l'écran du terminal. Ces caractères sont accompagnés d'un attribut qui en précise l'apparence : l'attribut de visualisation.

Représentation externe.

(ligne,colonne,texte)*
(ligne,colonne,longueur,code d'attribut)*

Exemple.

```
1 1 Entrez
1 10 le
1 14 nom
1 18 du
1 20 client
0 0 79 1234
1 0 1839 2456
```

Fonctions d'accès.

fill_ecra(car,ligne début,colonne début,ligne fin,colonne fin);
Rempli une portion rectangulaire d'écran à l'aide d'un caractère, sans en changer les attributs.

fill_att(car,ligne début,colonne début,ligne fin,colonne fin);
Modifie les attributs d'affichage d'une portion rectangulaire de l'écran.

move_ecra(ligne début,colonne début,ligne fin,colonne fin origine,
ligne début,colonne début destination);
Déplace une portion rectangulaire d'écran, accompagnée des attributs.

chg_attr(code attribut);
Cette fonction défini l'attribut "courant", qui sera utilisé par la fonction "put_ecra" définie ci-dessous.

put_ecra(ligne,colonne,chaine de caractère);
Place une chaine de caractère aux coordonnées ligne,colonne, en utilisant l'attribut courant.

aff_ecra(ligne début,colonne début,ligne fin,colonne fin);
Affiche une portion rectangulaire d'écran sur l'écran du terminal.

load_ecra(fichier,déplacement ligne,déplacement colonne);
save_ecra(fichier,déplacement ligne,déplacement colonne);
Ces fonctions, respectivement chargent et sauvegardent des dessins d'écran dans un fichier référencé par une variable "fichier" (Pointeur de fichier défini en C en tant que

FILE*). Dans le cas de la sauvegarde, les déplacements constituent des valeurs à ajouter aux coordonnées ligne,colonne présentes dans le fichier. Lors du chargement, on retranchera ces valeurs. Ces valeurs permettent d'obtenir des fichiers de dessins d'écrans indépendants de l'emplacement réels où ils se trouvent sur l'écran. Il suffit donc de modifier ces valeurs pour déplacer un dessin d'écran complet.

5.8 Les zones de saisie.

Comme nous l'avons déjà mentionné, les zones de saisies sont liées à des variables du programme.
Comme nous l'avons expliqué au paragraphe 1.2., l'analyse des besoins insiste sur la nécessité qu'a CSGen d'être le plus convivial possible, ce qui se traduit entre autre par l'adoption d'un maximum d'automatismes, résultant en un minimum d'efforts nécessaires à la réalisation d'un écran.

Définition.

Une zone de saisie est une portion rectangulaire située à un endroit défini de l'écran qui correspond à une variable du programme et qui pourra être affichée ou éditée par l'utilisateur. Dans un programme, une variable est identifiée par son nom. La taille de la portion rectangulaire défini la taille de la variable. Nous avons déjà abordé les différents attributs d'une zone de saisie au paragraphe 4.6.

Représentation externe.

```
( ligne début, colonne début, attribut de visualisation,
  Nom de la variable associée) *
```

Exemple.

```
1 1 1234 date
4 5 234 cli.nom
6 7 0 ligne(0)
```

Fonctions d'accès.

```
getvar(ligne,colonne);
```
Cette fonction fournit les attributs d'une variable affichée aux coordonnées (ligne,colonne) de l'écran, s'il en existe une. Sinon, un code d'erreur est renvoyé.

```
inpvar();
```
Cette fonction permet d'éditer les attributs d'une variable préalablement référencée par getvar.

```
putvar();
```

Suite logique des deux précédentes fonctions, putvar sauvegarde les attributs de la variable référencée préalablement par getvar.

```
aff_ecra(ligne début,colonne début,ligne fin,colonne fin);
```

Affiche sur l'écran du terminal la ou les zones de saisie définies sur une portion rectangulaire d'écran.

5.9 Les zones de fond.

Définition.

Une zone de fond détermine une portion rectangulaire située à un endroit défini de l'écran. Une zone de fond est identifiée par son nom et est destinée à recevoir un dessin d'écran et/ou des zones de saisie qui possèderont, par défaut des attributs de visualisation.

Représentation externe.

```
( Ligne début, colonne début, ligne fin, colonne fin,
  attribut par défaut du texte,
  attribut par défaut des zones de saisie,
  nom de la zone de fond )
```

Exemple.

```
0 0 2  79 1234 5678 Titre
3 0 18 79 3421 7654 Dialogue
19 0 19 79 4 67 Options
```

Fonctions d'accès.

Possèdant une structure proche de la structure des zones de saisie, les fonctions d'accès aux zones de fond sont similaires aux fonctions d'accès aux zones de saisie.

```
getzone(ligne,colonne);
inpzone();
putzone();
```

5.10 L'édition de l'écran de fond.

Un écran de fond est constitué des composants élémentaires suivants :

- Un dessin d'écran.
- Des zones de fond non superposées.
- Des zones de saisie de type "Out" non superposées entre elles et non superposées aux zones de fond.

L'édition de chacun de ces éléments peut avoir lieu sur l'entièreté de la surface de l'écran du terminal.

Nous avons développé, à l'aide des fonctions d'accès énoncées ci-avant, un éditeur d'écran qui prend en compte les composants utilisés par l'écran de fond.

5.11 L'édition des grilles d'écran.

Une grille d'écran est constituée des composants suivants :

- Un dessin d'écran.
- Des zones de saisie non superposées entre elles.

L'édition de chacun de ces éléments ne peut avoir lieu que dans les limites définies pour la zone de fond "Dialogues".

5.12 L'édition des écrans physiques.

Un écran physique est constitués des composants suivants :

- Une ou plusieurs grilles d'écran.
- Pour chacune des zones de saisie de ces grilles le "type" de la zone de saisie.
- Des dessins d'écran dans toutes les zones de fond éditables, excepté dans la zone de "dialogues".

Il y plusieurs méthodes pour mener à bien l'édition d'un écran physique, selon que l'on place toutes les grilles sur un pied d'égalité ou que l'on en privilégie une d'entre elles. Dans le premier cas, l'édition de l'écran physique se borne à superposer des grilles déjà définies. Dans le second cas, cette superposition est complètée par l'édition d'une grille d'écran particulière, tout en visualisant "en fond" les autres grilles. Cette seconde solution n'offre finalement que peu d'avantages au programmeur, elle a donc été rejetée. De plus, elle apporte un inconvénient par la confusion des concepts de grille d'écran et d'écran physique (Elle permet d'ailleurs de ne pas employer les grilles d'écran au même titre que CSGen utilitaire).

5.13 Edition du scénario.

Les composants du scénario d'une application ont été décrits aux paragraphes 4.9 et 4.10. L'édition du scénario est, chronologiquement, la première étape de la création des écrans d'une application. On aurait pu autoriser le développeur à définir des écrans n'apparaissant pas dans le scénario (Les écrans physiques étant finalement indépendants) de manière à assouplir sa tâche mais cette démarche risque de conduire à des incohérences que nous avons préféré éviter.
L'éditeur du scénario permet, à partir du nom d'un écran physique, de modifier la liste de ses touches de sortie et, pour chacune d'entre elles, de modifier les attributs de

l'enchaînement de l'écran physique suivant (Test, instruction d'appel de la fonction-arc et attribut de mise-à-jour).

5.14 Le compilateur d'écrans.

Le compilateur d'écran de CSGen superviseur, qui n'est pas encore développé, ressemblera de très près au compilateur de CSGen utilitaire. Le principe de génération de sources C sera conservé, nous l'avons déjà abordé au paragraphe 4.9. Comme pour CSGen utilitaire, la compilation des écrans consistera principalement en la numérotation des zones de saisies et en la production de fichiers-écrans optimisés, de manière à réduire les temps d'accès lors de l'exécution de l'application.
La génération des sources contenant les définitions des variables sera plus complète. Elle séparera les fichiers liés aux vecteurs, aux matrices et aux structures, et générera ces références en "external". Il sera également possible de générer des sources à partir d'un schéma défini par l'utilisateur, pour obtenir, par exemple des schémas de définition de bases de données.

5.15 La librairie d'exécution.

La librairie d'exécution de CSGen superviseur contiendra les fonctions d'exploitation des fichiers-écrans, c'est-à-dire la lecture et l'affichage "différentiel". Les autres fonctions de la librairie permettront les enchaînements entre les écrans physiques, via les fonctions arcs. Les fonctionnalités fournies lors de l'édition d'un écran seront semblables, les améliorations (tant au niveau de la programmation qu'éventuellement au niveau de l'interface utilisateur) résulteront de l'adjonction des fonctions à priori et à postériori et du "buffer des zones de saisie".

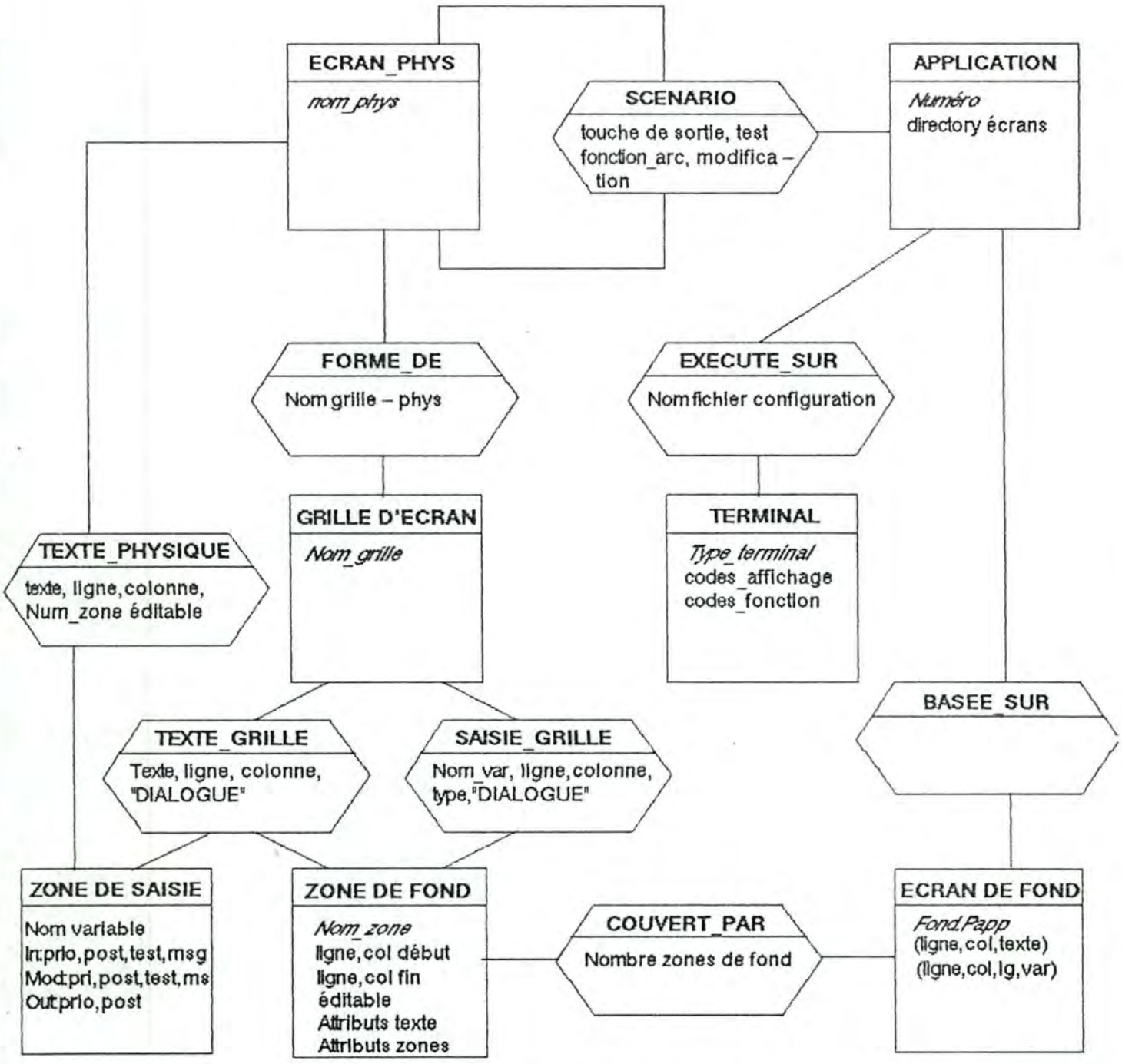
5.16 Le programme d'installation.

Le programme d'installation permet à l'utilisateur d'adapter CSGen à l'environnement matériel et organisationnel.
Le but principal du programme d'installation est de définir le terminal de l'utilisateur, c'est-à-dire de choisir les paramètres du terminal utilisé parmi une liste de terminaux supportés par CSGen ou bien de permettre à l'utilisateur de définir un terminal qui n'existe pas dans la liste.
Le programme d'installation permet également au développeur de décider des "directory" qui contiendront d'une part les fichiers-écrans et les programmes source de l'application et d'autre part les programmes de CSGen (L'éditeur et le compilateur d'écrans).
Il est aussi possible, à l'aide de ce programme, de définir les noms d'options par défaut attachés aux touches de sortie d'écran qui ont été définis au paragraphe 4.4.

5.17 Le schéma entité-association de CSGen.

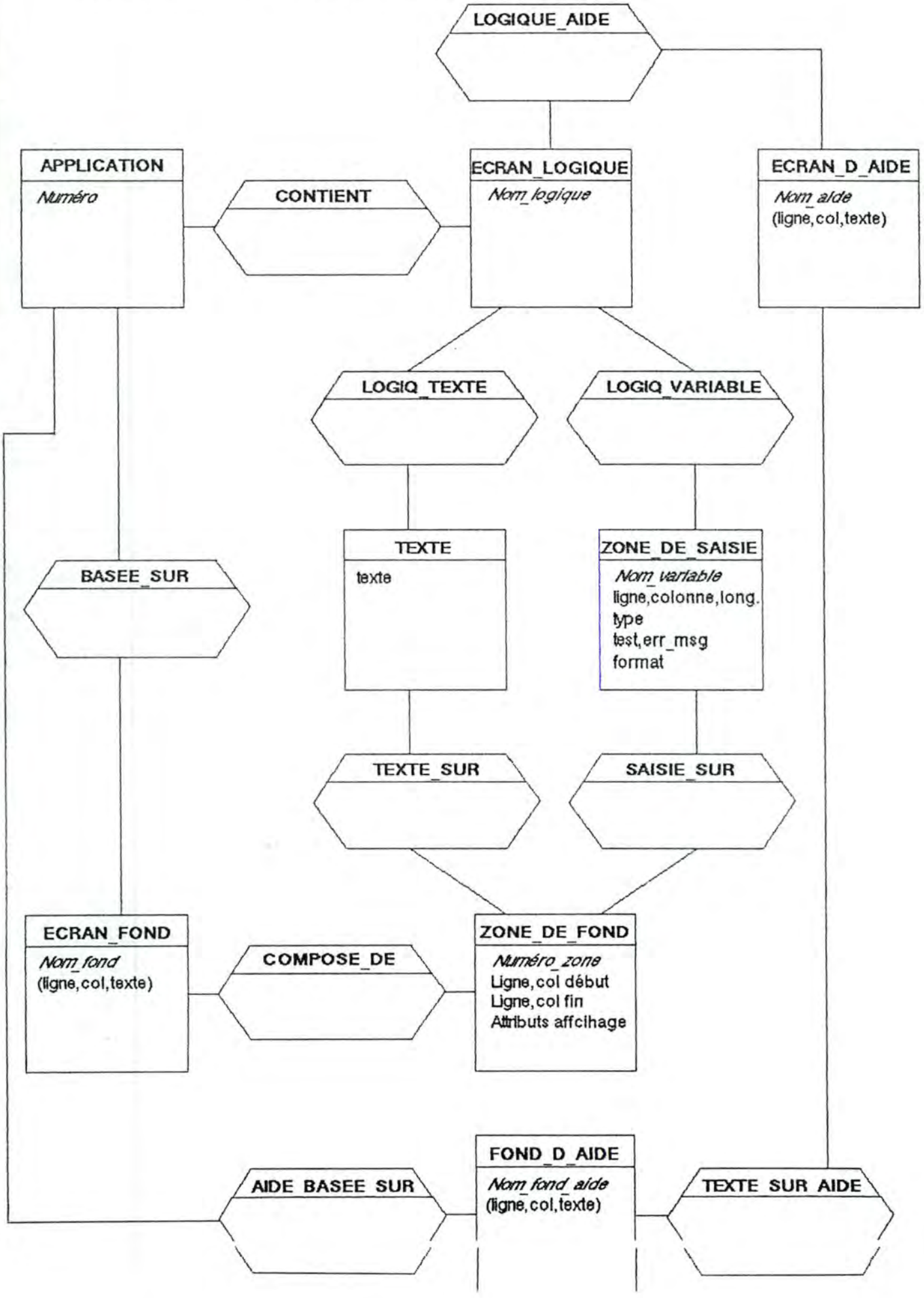
Les différents composants définis ci-avant peuvent être organisés selon le schéma entité-association illustré à la figure 5.1.
La figure 5.2. illustre le schéma entité-association de CSGen utilitaire qui n'est pas commenté et qui n'est représenté qu'à des fins de comparaisons.

FIGURE 5.1. Schéma E-A de CSGen Superviseur.



texte = aggrégat formé d'une chaine de caractère et des attributs de visualisation.

FIGURE 5.2. Schéma E-A de CSGen Utilitaire.



CHAPITRE 6 : EXTENSIONS DE CSGEN SUPERVISEUR.

6.1 La gestion de la souris.

On peut affirmer que la souris est un interface dont l'emploi se généralise rapidement dans le domaine grand public.
Les routines d'édition d'un écran peuvent facilement être adaptées pour utiliser la souris et permettre à l'utilisateur et au programmeur d'accéder rapidement à n'importe quel endroit de l'écran.
Les difficultés présentées par l'utilisation de la souris sont liées à deux techniques qu'elle a introduites : Les menus déroulants (dont un exemple est représenté à la figure 6.1.) et la disparition des touches de sortie au profit d'icônes (voir figure 6.2.). En effet, dans ce cas l'acceptation d'un écran est généralement réalisée de deux manières :

- La sélection d'un icône d'acceptation.

  On peut considérer qu'il ne s'agit ici que d'une nouvelle forme de touche de sortie, l'adaptation de CSGen ne pose aucun problème car les systèmes d'exploitation fournissent un ensemble complet d'interfaces souris.

- Le choix d'un point de menu.

  Les applications basées sur l'utilisation de la souris affichent généralement en permanence le menu principal. La sélection d'un point de menu peut donc intervenir à n'importe quel moment au cours du dialogue homme - machine. Le choix d'un point de menu doit alors être considéré par CSGen comme l'appui sur une touche de fonction inconnue, et demander à l'utilisateur s'il veut conserver les modifications ou non, puisque dans ce cas l'attribut "mise-à-jour" est inconnu.

FIGURE 6.1. Exemple de menu déroulant.

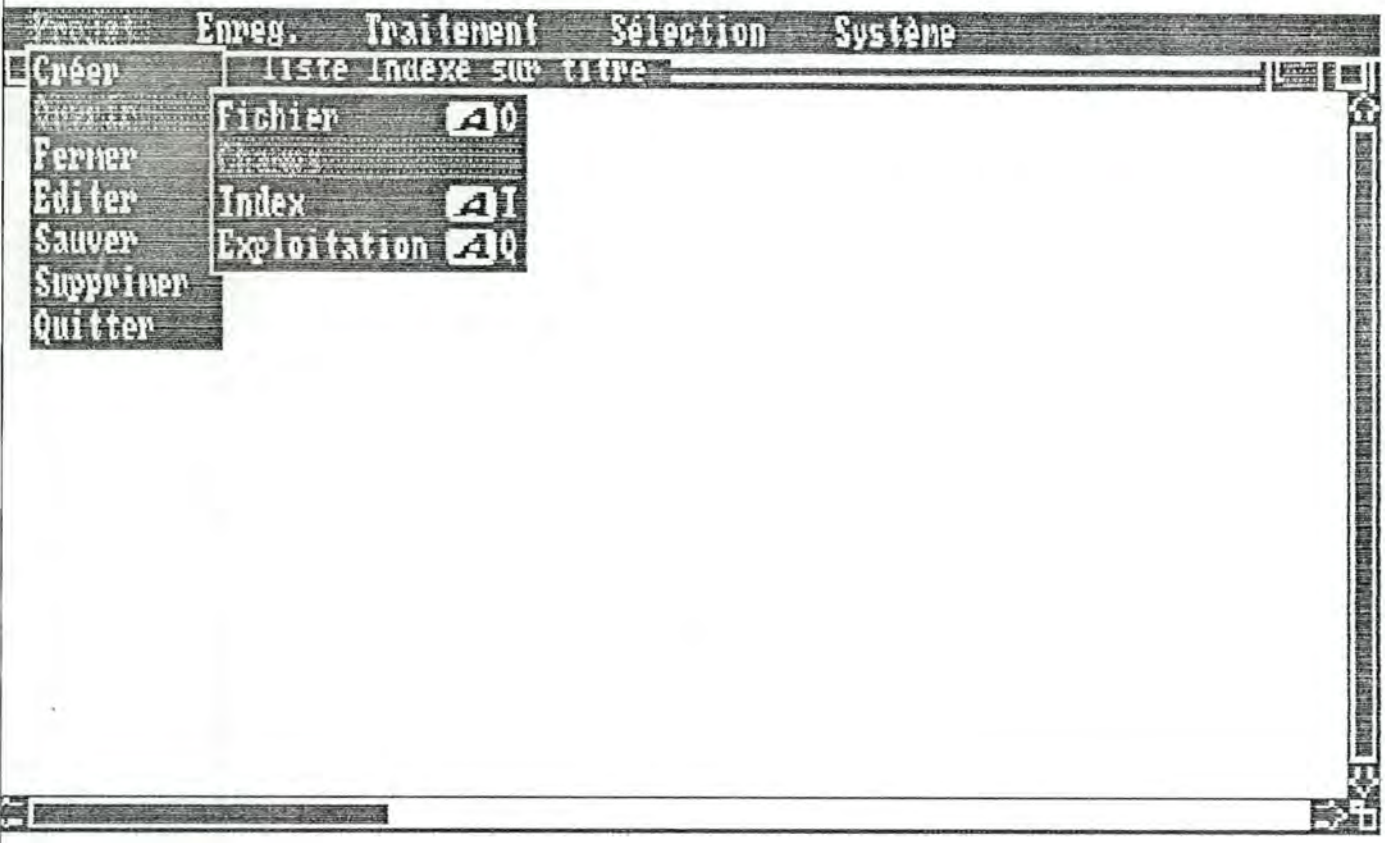
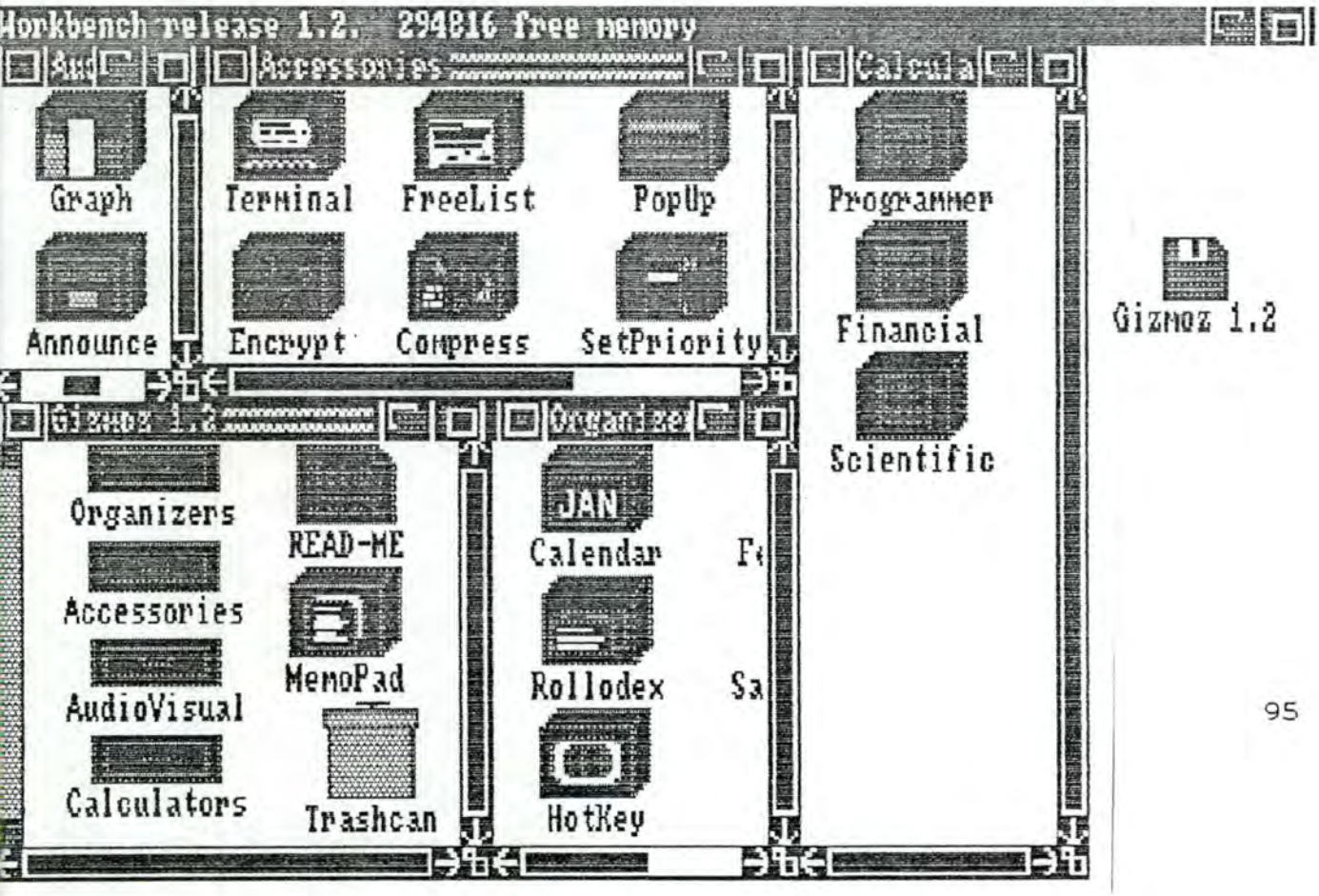


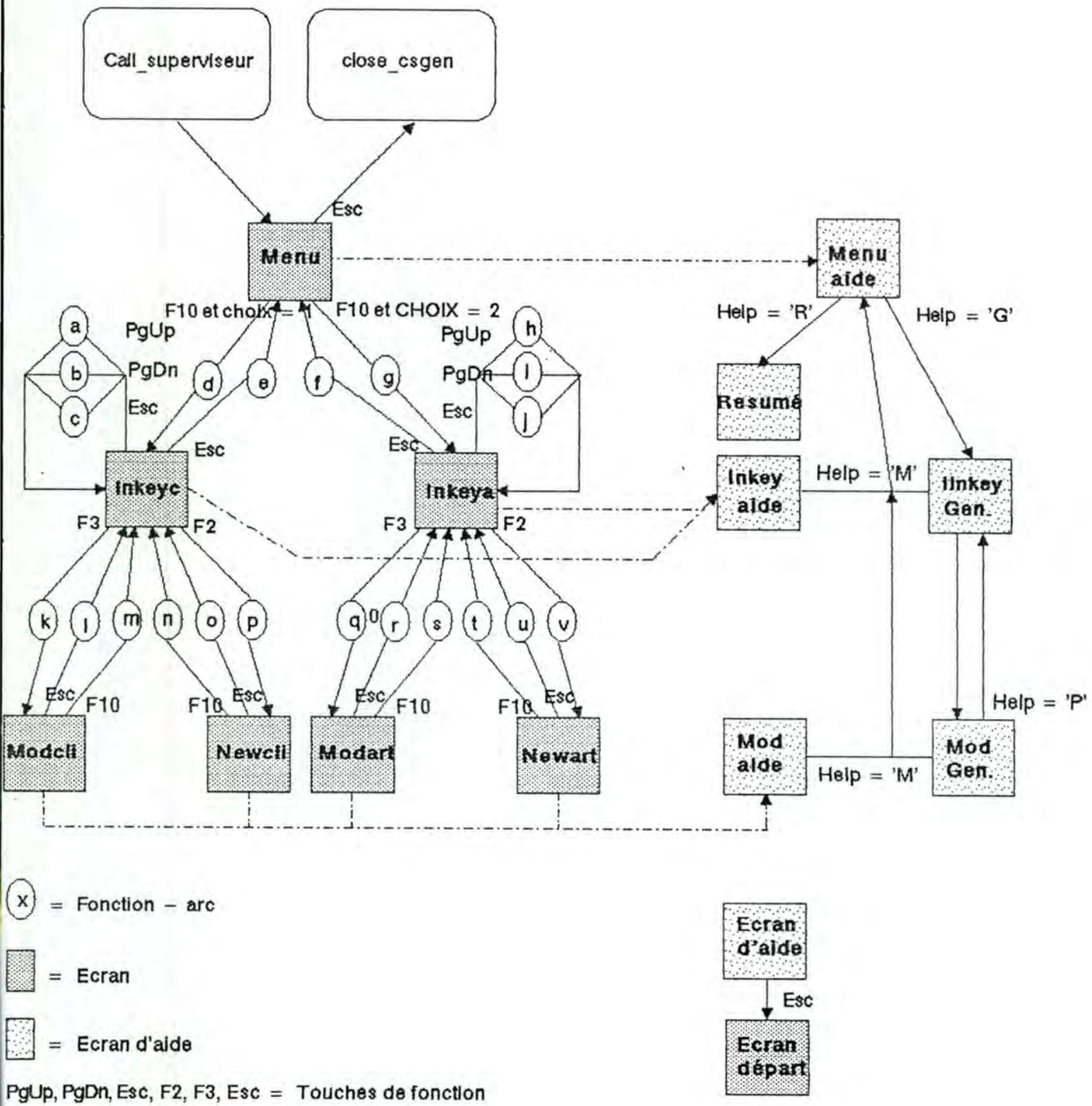
FIGURE 6.2. Exemple d'icône.



6.2 Amélioration des écrans d'aide.

Nous avons volontairement passé sous silence les écrans d'aide de CSGen superviseur car leur gestion n'est actuellement envisagée qu'en tant qu'extension.
Le système d'écrans d'aide utilisé par CSGen utilitaire est lourd a développer, il faut en effet créer un écran d'aide pour chaque écran logique. Nous envisageons de permettre la définition d'un scénario d'aide et d'utiliser des zones de saisies dans les écrans d'aide. Cette méthode permet d'inclure des points de décision dans les écrans d'aide, ce qui est utile lorsqu'il en existe un grand nombre (par exemple le "guide" de WORD3 ou l' "assist" de Dbase3). Dans cette approche, les écrans d'aide constituent une forme simplifiée d'écrans de dialogue, on ne parlera pas dans ce cas de grilles d'écran ou de fonctions à postériori. Le scénario d'aide ne comprendra pas de fonctions-arc et on peut raisonnablement limiter à deux le nombre de touches de sortie d'écran : Une touche qui permet "d'avancer" dans l'aide et une autre qui permet le retour à l'écran de dialogue suspendu. Lors d'appuis sur la touche d'avancement, les aiguillages entre les différents écrans d'aide pourront être réalisés à l'aide du "test de sortie" et des zones de saisie. La figure 6.3. illustre le scénario de la seconde solution (énoncée au paragraphe 2.9) accompagné du scénario d'aide.

FIGURE 6.3. Exemple de scénario d'aide.



6.3 Vers un langage de quatrième génération.

Cette extension de CSGen résulte de deux constations :

-Les écrans développés avec CSGen contiennent beaucoup d'informations qui seront insérées dans les sources de l'application. Ces informations constitueront le programme, au même titre que les fonctions développées par le programmeur.

-Les sources des fonctions-arc seront, dans la majorité des cas, constituées d'un petit nombre d'instructions.

Il serait donc intéressant de considérer les sources des fonctions-arc comme des informations liées aux écrans physiques. En effet, si ces informations sont liées logiquement aux écrans, elles devraient l'être également physiquement. CSGen deviendrait alors l'unique outil nécessaire à l'écriture d'un programme, intégrant un éditeur d'écrans et un éditeur de textes. Cette approche offre l'avantage de supprimer les problèmes de morcellement des programmes évoqués au paragraphe 4.11. Il est de plus relativement facile d'ajouter un éditeur de texte à CSGen, puisque la plupart des fonctions nécessaires à l'édition existent déjà dans l'éditeur d'écran.

Cette démarche pourrait être complétée par un dispositif simplifiant la programmation qui, de ce fait, permettrait un accroissement de la rentabilité des programmeurs ou encore permettrait à des non informaticiens de créer leurs propres programmes.

Pour mener à bien cette tâche, CSGen peut fournir des macros-instructions de très haut niveau fournies sous la forme d'une librairie ou d'un interpréteur. Cette seconde approche pourrait éviter au programmeur de devoir se plier à la syntaxe du langage C qui n'est pas très "digeste" pour un débutant.

On peut aussi envisager l'adjonction d'un générateur de programme qui servirait d'interface "évolué" entre CSGen et l'utilisateur. Ces extensions sont réalistes puisque CSGen possède déjà toutes les fonctions de manipulation d'écrans et de leurs variables ainsi que des fonctions d'accès aux fichiers indexés.

CHAPITRE 7 : CONCLUSION.

La nécessité d'employer des utilitaires adaptés n'est plus à démontrer.
Tout le problème réside dans le mot "adapté" !

Nous avons dépeint deux utilitaires qui remplissent les mêmes fonctions, bien qu'ils différent totalement par leur implémentation et par leur utilisation.

Le premier utilitaire (CSGen utilitaire) apporte peu de contraintes au programmeur, il s'agit réellement d'un outil de programmation au même titre qu'un gestionnaire de fichier. Par contre, cette approche a prouvé qu'elle impose des limitations et des complications, dont certaines sont paradoxalement dues au désir de ne pas contraindre le programmeur.

Au contraire, le second utilitaire (CSGen superviseur) est très contraignant puisqu'il impose sa propre découpe du programme. Il en découle la nécessité d'une analyse adaptée. Néanmoins, cette démarche apporte une grande simplification de la programmation en supprimant toute référence explicite aux écrans et à leurs enchaînements et en obligeant le développeur à se concentrer sur les fonctions-arcs qui sont des traitements élémentaires.

Comme c'est souvent le cas, on ne peut pas parler de panacée universelle. La raison en est probablement que les langages informatiques couramment employés ne sont pas centrés sur le dialogue homme-machine. Il faut alors plutôt se rabattre sur des langages tels que OMEGA ou sur des générateurs d'applications.

BIBLIOGRAPHIE.

"L'informatique conversationnelle", B.Faulle, .

"Le langage C", Ritchie & Kernighan,

"Micrac : Manuel technique", I.T.C., 1985.

"ICL 6402G User guide", I.C.L., 1985.

"Peripherals configuration, dealer course", I.C.L., 1986.

"GST-C User Manual", GST Edit, 1985.

"Turbo-C Reference Guide", Borland Inc., 1987.

"Lattice C User Manual", Lattice Inc., 1984.