INTERET, CREATION ET UTILISATION DE GRILLES D'ECRAN

Patrick Séchet
(Informatique)
(UR 502, Cadres Spatiaux de l'Indépendance Alimentaire)

EMBRAPA
Empresa Brasileira de Pesquisa Agropecuária
SuperCenter Venâncio 2.000
70333 - Brasília, DF (Brésil)

RESUME - En matière de langage, on a le choix entre deux possibilités pour la mise au point d'une application sur micro-ordinateur : adopter un progiciel spécifique de gestion de données ou utiliser un langage de programmation.

La première option est très séduisante, surtout parce qu'elle permet de disposer immédiatement de diverses ressources comme la saisie formatée, la génération d'états, les menus d'aide, etc.. Toutefois, on finit toujours par se heurter à telle ou telle restriction inhérente à la méthode d'accès, à la structure des données, aux possibilités de calculs mathématiques, à la capacité de génération de graphiques, à l'utilisation de tables de codification-décodification, etc..

D'un autre côté, en utilisant un langage de programmation on souffre de l'absence de ressources satisfaisantes pour l'entrée des informations. Cette note présente la réalisation d'un éditeur et gestionnaire de grilles d'écran qui favorise le dialogue entre l'utilisateur et le système, en facilitant la saisie des données.

Les versions les plus récentes, opérationnelles sur micro-ordinateurs CP/M 80 et MS-DOS, ont été développées en langage Pascal par l'équipe franco-brésilienne du programme SISGEO : les commandes de manipulation du curseur, édition de texte et saisie sont conformes au standard introduit par le progiciel de traitement de texte *"Wordstar"*.

Cet outil est constitué de deux éléments distincts : un éditeur pour dessiner les grilles d'écran et un sous-programme d'accès à ces écrans, le gestionnaire d'écrans.

L'éditeur de grilles d'écran est un programme qui produit et manipule des fichiers normalisés qui contiennent l'image et la définition des grilles qui seront utilisées par les programmes applicatifs. Durant une session d'édition, on a toutes les facilités d'un

traitement de textes pour dessiner la grille sur l'écran, telle qu'elle est souhaitée par le développeur.

Chaque caractère de la grille est soit une constante, soit une variable, non connue au moment du dessin de l'écran et que l'on représente par un caractère spécial. On distingue dans ce dernier groupe les champs accessibles par l'utilisateur (options, données captées) des champs protégés, non accessibles.

Ces derniers peuvent être élaborés à partir d'informations disponibles ou simplement obtenus par l'intermédiaire d'une grille précédente. Ils sont restitués, du système à l'utilisateur, par l'intermédiaire de la grille d'écran (date de session, par exemple).

Un ensemble d'options de l'éditeur fournit les fonctions nécessaires à la manipulation de ces fichiers. En particulier, l'édition des grilles est permise sous deux formes différentes :

- sous forme de grille d'écran simple (image de ce qui sera présenté sur le moniteur), utilisée pour le dialogue avec le commanditaire du travail et dans la documentation du système ;
- sous forme de grille d'écran accompagnée d'un dictionnaire des variables correspondantes (numéro, position sur l'écran et format d'édition de chaque champ variable), utilisé par le programmeur pour la codification.

Le gestionnaire de grilles d'écran est une subroutine appelée par les programmes applicatifs et sert d'interface avec les écrans. Il fournit essentiellement trois fonctions :

- affichage d'une grille d'écran, ce qui suppose l'ouverture et la lecture d'un fichier correspondant (ou la recherche dans une bibliothèque d'écrans), et la visualisation proprement dite ;
- écriture d'une valeur dans le champ variable correspondant, protégé ou non, destiné à informer l'utilisateur du système ;
- récupération de l'information fournie par l'opérateur à partir de la grille d'écran et mise à disposition du programme hôte.

L'utilisateur dispose pour la saisie de toutes les capacités d'édition *"plein écran"*. Trois types de champs sont définis : numérique fixe, numérique *"flottant"* et alphanumérique.

Le transfert des informations saisies avec le programme hôte se fait à chaque champ obtenu, ce qui apporte un maximum de souplesse pour la validation, qui doit être codifiée dans le programme principal.

L'adoption d'un outil de ce type est de nature à optimiser la mise au point des systèmes applicatifs qui font intervenir de la saisie de données sur écrans formatés. Il favorise en outre la normalisation des programmes produits (donc la maintenance ultérieure) et standardise également l'interface avec les utilisateurs de ces

programmes. Enfin, un avantage (et non des moindres) est qu'il génère sa propre part de documentation.

1. JUSTIFICATION

L'aspect conversationnel de la relation entre l'homme et la machine est, sans aucun doute, l'un des avantages prépondérants que la micro-informatique a apporté par rapport à l'informatique traditionnelle. Cet aspect est particulièrement sensible à la simplicité syntaxique du dialogue avec l'ordinateur et à la brièveté du temps de réponse de celui-ci, qualités que l'on ne rencontre pas toujours lors de l'utilisation de ressources centrales partagées.

Ce dialogue s'établit, par l'intermédiaire du clavier et de l'écran, et se compose d'une succession d'informations, tantôt fournies par l'ordinateur, tantôt par l'opérateur.

En utilisant les ressources de base des langages de programmation, un échange de données en temps d'exécution pourra être effectué, une ligne après l'autre, par une séquence d'ordres de lecture et d'écriture à l'écran (dans la plupart des langages : Read pour une information à fournir par l'opérateur, Write pour une réponse donnée par le système).

Le dialogue créé de cette façon est assez précaire. En effet, lorsque l'écran est plein, l'affichage d'une nouvelle ligne provoque une déplacement vers le haut (*"scroll-up"*) de l'ensemble des informations présentes sur l'écran, la ligne située au sommet disparaissant. Il en résulte une perception séquentielle et partielle, la plupart du temps non satisfaisante, des informations échangées. De plus, les commandes d'édition de champs dont on dispose pour recevoir des informations de la part de l'utilisateur du système sont rudimentaires : le plus souvent seule la touche (*"backspace"*) de retour arrière avec effacement est opérationnelle pour une instruction read, permettant la correction d'une entrée erronée.

Pour améliorer ce système de communication, on a d'abord imaginé un dialogue par le biais d'écrans pleins et, plus récemment, de portions d'écran (fenêtres), en considérant un écran comme un module capable de véhiculer les informations, dans les deux sens, de forme tout à fait semblable à un formulaire. Les écrans servent alors à présenter des informations à un utilisateur, à lui poser des questions, lui présenter une suite d'options, fournir une grille de saisie, etc..

Ceci constitue un moyen efficace de structurer le dialogue, dans la mesure où l'on projette chaque fenêtre pour qu'elle contienne des données appartenant à un même groupe, exactement comme l'on dessine un gabarit de rapport ou le *"lay-out"* d'un bordereau.

Ce type de dialogue structuré est adopté par un grand nombre de progiciels de toute nature : il est généralement facile à réaliser avec les langages de prototypage, ce qui est une des raisons pour lesquelles beaucoup de systèmes sont mis au point avec ces derniers.

Parmi les applications qui tirent le plus grand bénéfice de cette manière de travailler on peut distinguer tous les systèmes destinés à la capitalisation d'information, qui utilisent les écrans pour la saisie des données : en particulier les systèmes destinés au dépouillement et traitement des enquêtes. Bien entendu, la réalisation du dialogue système-utilisateur par le biais de fenêtres peut être introduit pour les consultations structurées, la mise à jour d'informations, l'aide en ligne, etc., jusqu'aux simples écrans de menus, de telle façon qu'une application simple sur micro-ordinateur pourra devoir administrer plusieurs fenêtres, allant souvent jusqu'à quelques dizaines.

2. AVANTAGES

Il est bien sûr tout à fait possible de mettre au point, de forme indépendante dans chaque programme d'application, les ressources de visualisation et récupération d'information par le biais de fenêtres. Toutefois, l'utilisation d'un module spécifique d'administration de ces écrans apporte un certain nombre de bénéfices et permet de satisfaire à plusieurs objectifs dans le contexte de développement de progiciels.

Telle méthode permet en effet :

- *d'accélérer la mise au point.* Une fois que le programmeur domine cet outil, toutes les fonctions d'entrée-sortie par écran des programmes sont facilement et rapidement codifiées. Il est très difficile d'évaluer le gain de temps qui peut être apporté durant cette phase, compte tenu de la diversité des applications possibles. On peut néanmoins signaler qu'il existe toujours une interface de communication entre le progiciel et son utilisateur (par conséquent toutes les applications sont concernées). Celle-ci constitue parfois l'essentiel du système, par exemple lorsque le but poursuivi est de stocker des données : dans ce dernier cas l'effort de programmation pourra être réduit substantiellement par l'utilisation d'un gestionnaire d'écrans ;

- *de réduire la taille des programmes.* Fenêtres et écrans sont en effet traités comme des fichiers externes aux programmes, de sorte que la taille de ces derniers est réduite, aussi bien en ce qui concerne le code source que le module objet. Ces fichiers peuvent d'ailleurs être partagés par plusieurs programmes (écrans d'aide, par exemple), ce qui constitue aussi un facteur de réduction de la

taille des programmes, tout en évitant des redondances. De plus, le traitement d'un écran est normalement réalisé avec quelques lignes de code (appels de procédures ou fonctions), en remplacement de toute une succession d'ordres de lecture et d'écriture de la méthode conventionnelle. Enfin, un gestionnaire d'écrans pourra parfaitement se charger de l'essentiel de la validation des informations fournies par l'utilisateur, dispensant le programme applicatif d'encombrantes routines de validation et manipulation d'erreurs. L'avantage de réduction de la taille des programmes est particulièrement significatif lorsque la mémoire interne est limitée : dans tous les cas, le travail de programmation est réduit, non seulement parce qu'il y a moins de code, mais aussi parce que celui-ci est mieux structuré et plus simple ;

- *de promouvoir l'indépendance* entre le dialogue et les programmes. Toute modification de la communication entre le progiciel et son utilisateur se traduira par une mise à jour de l'écran correspondant, sans nécessairement avoir recours à une modification de programme. Alors que dans un environnement traditionnel il est nécessaire de rechercher et modifier plusieurs ordres de lecture et d'écriture dans un programme au moins ; si l'on utilise un gestionnaire d'écrans, il suffira d'éditer l'écran à modifier pour réaliser les changements voulus ;

- *de normaliser les programmes.* La communication entre les programmes et le moniteur utilise un unique intermédiaire, le gestionnaire d'écrans. Celui-ci étant le même pour tous les programmes, même développés par des personnes différentes, dans le cadre d'un même système applicatif, il s'ensuit que les tests et la maintenance s'en trouvent grandement facilités ;

- *de standardiser l'interface avec l'utilisateur.* Ce dernier, en communiquant avec le progiciel, utilise normalement plusieurs programmes, sans forcément le savoir. Il lui est alors difficilement compréhensible que l'appui d'une même touche puisse avoir des effets différents, lors de phases distinctes du traitement. L'utilisation du gestionnaire (Figure 1) est une garantie de ce que l'utilisateur peut attendre un comportement unique de toutes les ressources du progiciel, en ce qui concerne la fonction des touches, les facilités d'entrée de données, la présentation des messages, etc. ;

- *de favoriser la maintenabilité.* Les ordres de lecture et écriture, ajoutés aux routines de validation, occupent une partie significative de beaucoup de programmes : le fait que l'ensemble puisse être substitué par quelques appels de fonctions contribue à faciliter tests et maintenance. Par ailleurs, la partie de code correspondant au dialogue est *"mise en facteur"* : ainsi, la visualisation d'une fenêtre à l'écran s'écrit sous la forme d'une commande unique. Ceci améliore la lisibilité du programme, et, partant, sa

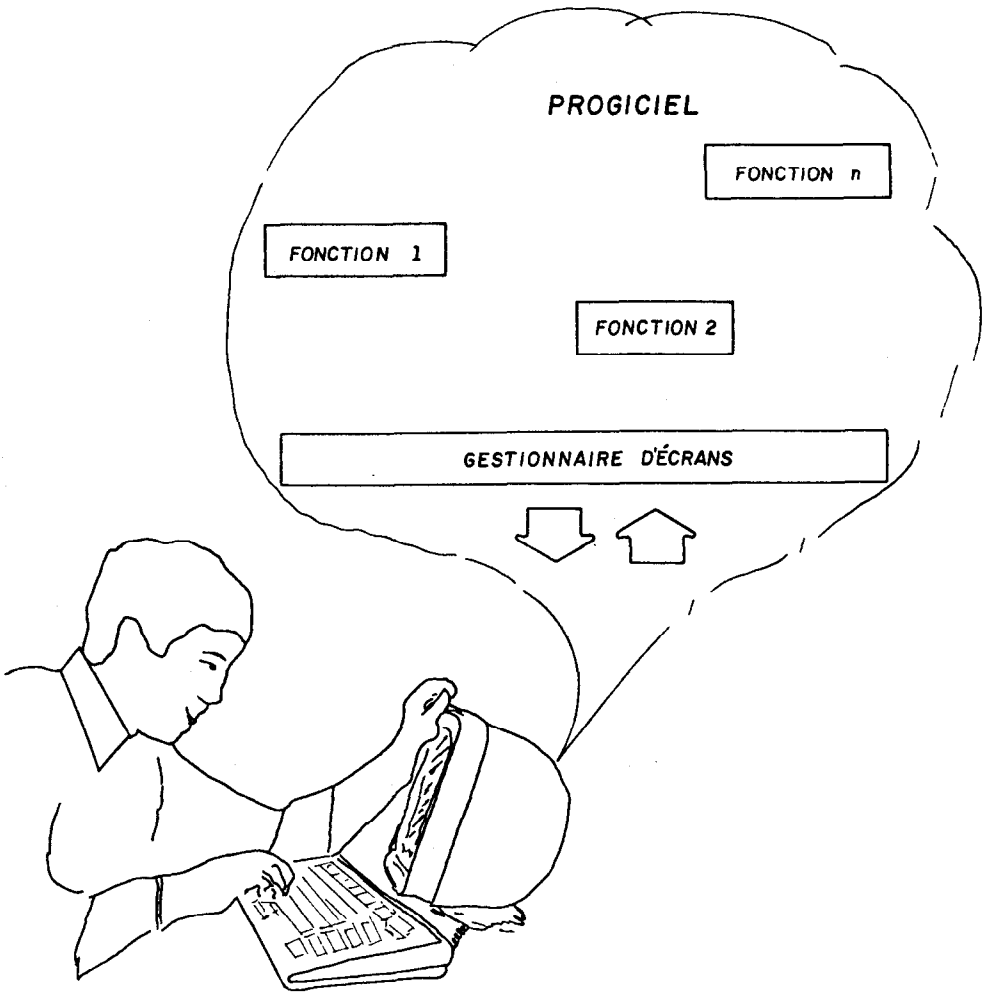


Figure 1 - Le gestionnaire d'écrans administre totalement la communication entre le progiciel et l'utilisateur.

maintenabilité. La portabilité de l'application augmente dans les mêmes proportions ;

- *d'automatiser une partie de la documentation.* Tout logiciel de création et utilisation d'écrans permet l'impression de ceux-ci, généralement sous deux formes distinctes. La première fournit l'image de l'écran tel qu'il sera visualisé au moment de l'exécution : au même titre que les lay-out de formulaires et les gabarits d'états, elle pourra faire partie de la documentation des entrées-sorties du progiciel, soumis à approbation du commanditaire avant de débuter la programmation. La seconde comporte en plus toutes les informations nécessaires sur le format et la position de chaque champ contenu dans l'écran en question : elle sera précieuse pour le programmeur chargé de la mise au point du programme qui l'utilise ;

- *d'améliorer le dessin des écrans.* Lorsque l'on ne dispose pas de cet outil, on doit dessiner les écrans sur un gabarit, de façon à déterminer la position (ligne et colonne) de chaque information et codifier les ordres de lecture et écriture correspondants. L'éditeur fournit avec le gestionnaire permet de dessiner les écrans interactivement, le plus souvent sous la forme selon laquelle ils apparaîtront en temps d'exécution, sans avoir à se soucier des positions des divers champs. Les produits les plus sophistiqués permettent d'améliorer la convivialité du progiciel, en tirant le meilleur parti des ressources offertes par les moniteurs : utilisation des attributs-vidéos monochromatiques (surbrillance, vidéo inversée, clignotement), des caractères spéciaux et semi-graphiques de l'IBM-PC (cadres, par exemple) ou des couleurs. Des efforts de programmation importants seraient nécessaires ou des couleurs. Des efforts de programmation importants seraient nécessaires pour atteindre un tel objectif sans l'utilisation de l'éditeur d'écrans ;

- *de viabiliser la séparation des tâches* d'analyse et de programmation. Lors de la mise au point d'un progiciel, le dessin des écrans peut être entièrement réalisé par l'analyste de spécification, indépendamment de la codification des programmes et de leur logique interne : ceci aura certainement pour effet d'améliorer le standard et le niveau de convivialité de l'application. En outre, la validation des données saisies peut également être prise en charge, au moins partiellement, par le gestionnaire d'écrans. Dans ces conditions, l'utilisation d'un tel produit s'accompagne d'un transfert substantiel d'activités de la phase de programmation vers la phase de spécification, ce qui est hautement bénéfique ;

- *d'optimiser les temps de réponse.* Dans la mesure où l'on utilise un produit spécifiquement conçu pour la manipulation des fenêtres et écrans, on peut espérer que les opérations correspondantes seront réalisées avec un niveau d'efficacité qu'il serait

difficile d'atteindre dans les conditions normales de développement d'application (langage de haut niveau, en particulier).

3. DESCRIPTION GENERALE

Pour la description générale des ressources d'un gestionnaire d'écrans, on est amené à définir trois contextes différents, liés au cycle de vie du système mis au point : la spécification, la programmation et l'exécution.

Dans le premier environnement, l'analyste utilise le produit pour dessiner les écrans et fenêtres qui interviendront dans le système applicatif projeté. Le gestionnaire est alors perçu comme un éditeur d'écrans, qui devra donc posséder des outils semblables à un éditeur de textes, tels que saisie de texte, mouvements du curseur, commandes de bloc, effacement, etc..

On pourrait à la limite se satisfaire d'un éditeur de textes standard (*Wordstar, Word, Wordperfect, Sprint,* etc.) pour réaliser les écrans. Cette solution apporterait toutefois quelques limitations dans la mesure où il faut créer une information qui ne peut pas être facilement véhiculée par le seul écran, comme les noms de champs, leurs types, leurs contenus par défaut, les valeurs initiales, minimales et maximales des champs numériques, etc..

Bien entendu, à l'instar des éditeurs de texte, l'éditeur d'écrans devra permettre de sauvegarder le résultat du travail sur disque (Figure 2), de le récupérer ultérieurement pour l'altérer si nécessaire, de l'exclure, de copier un écran pour en créer un nouveau, etc.. L'utilisation d'une bibliothèque de fenêtres et d'écrans pourra constituer une ressource intéressante, certains systèmes d'exploitation imposant une limite au nombre de fichiers contenus dans un volume ou dans une partition.

Enfin, l'éditeur devra permettre l'impression des écrans, sous les deux formes qui ont déjà été évoquées pour servir de documentation au programmeur et à l'utilisateur. La possibilité de créer un fichier texte contenant l'image de cette édition est appréciable pour permettre l'incorporation de cette documentation dans le cahier du programme et dans le manuel de l'utilisateur, par l'intermédiaire de l'utilisation d'un traitement de texte.

Dans le deuxième environnement, les programmes doivent être codifiés pour établir une interaction avec les écrans créés, qui se présentent désormais sous forme de fichiers sur disque, pour assurer le fonctionnement du progiciel.

Il faut alors disposer d'une bibliothèque de procédures ou de fonctions capables d'assurer, par simple appel en temps d'exécution, des tâches telles que la visualisation d'un écran ou d'une fenêtre, l'exhibition d'une information connue du système à

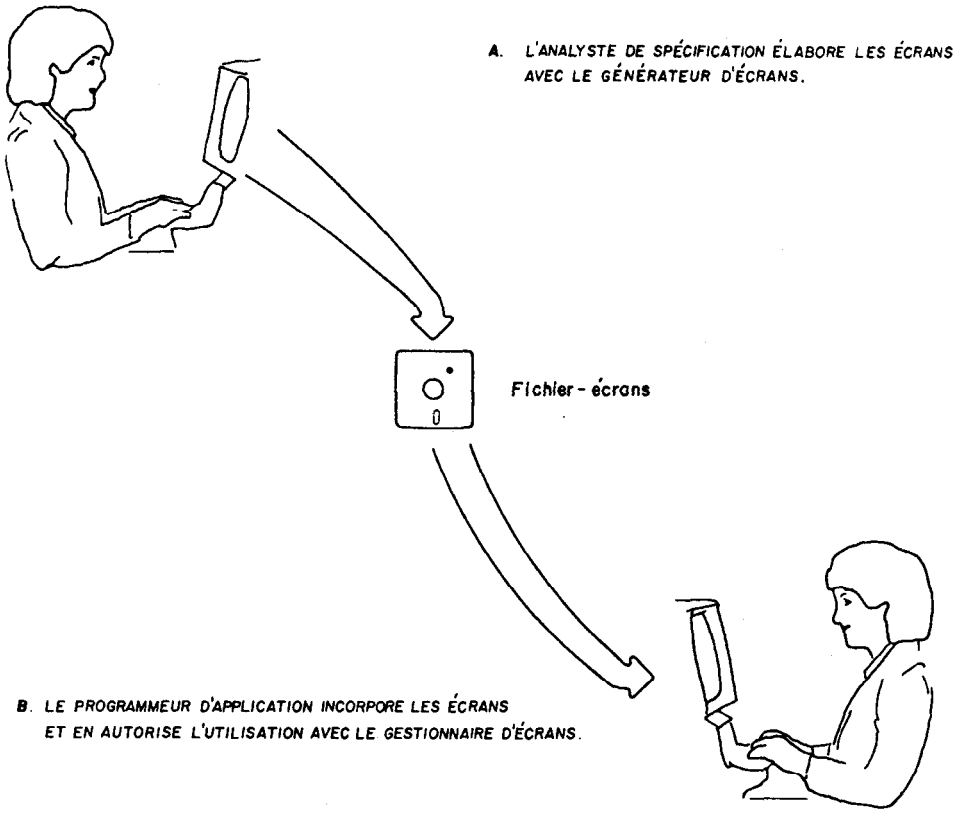


Figure 2 - L'utilisation d'un gestionnaire/générateur d'écrans met en évidence deux étapes distinctes.

l'intention de l'utilisateur (résultat d'un calcul, par exemple) ou encore la mise à disposition du progiciel d'une donnée fournie par l'opérateur.

L'interaction entre le programme d'application et les routines du gestionnaire d'écrans peut être réalisée de diverses manières, selon le langage et le produit utilisé. Dans le cas d'une version compilée d'un langage disposant de ressources d'appels de fonction et procédure d'une part, passant par une phase d'édition de liens d'autre part, le plus efficace sera de disposer des modules objets *relocables*, sous forme d'une bibliothèque complémentaire à concaténer avec la librairie standard du langage. Les routines du gestionnaire d'écrans seront alors automatiquement inclues dans le programme d'application.

Si l'on ne passe pas par une phase d'édition de liens (Turbo Pascal, par exemple), on préferera un produit composé d'un programme exécutable, chargé avant l'exécution dans un segment de mémoire distinct du segment de code et du segment de données (système d'exploitation MS-DOS). De cette façon, on aura les 64K du segment de code entièrement disponible pour le programme applicatif. Une autre solution, moins efficace, consiste à incorporer les sous-programmes du gestionnaire dans le programme source, puis compiler l'ensemble : l'inconvénient majeur de cette dernière solution est l'augmentation de la taille du programme objet, qui intègre alors toutes les routines (il y a néanmoins la possibilité d'utiliser le mécanisme de recouvrement).

Le dernier environnement est celui de l'utilisation effective de l'application développée, quand l'opérateur exécute les programmes qui composent le progiciel, et, par le biais de ceux-ci, manipule implicitement les fenêtres et écrans créés par le gestionnaire.

4. DESCRIPTION DE "EDITOR/TELADOR"

Le système d'administration d'écrans mis au point par l'équipe franco-brésilienne du programme SISGEO entre dans la catégorie des produits dont la description font l'objet de cette note. Il est utilisable en Turbo Pascal pour les systèmes d'exploitation CP/M 80 et MS-DOS. Rudimentaire, il convient bien pour servir d'exemple et illustrer dans le détail ces propos.

Comme son nom l'indique, il se compose de deux éléments distincts : un éditeur pour dessiner les écrans (telas, en portugais) et une bibliothèque d'accès appelée *"telador"*.

4.1. Editor

L'éditeur est un programme qui permet la création et la maintenance de fichiers standardisés, contenant l'image et la définition des écrans qui seront utilisés dans l'application en cours de mise au point.

Pendant une session d'édition, on dispose de toutes les facilités pour dessiner la grille d'écran sur le moniteur : la définition des champs qui seront considérés comme variables (c'est à dire non connus lorsque l'on construit la grille, mais seulement déterminés en temps d'exécution du programme applicatif) est faite grâce aux caractères spéciaux placés sur la grille elle-même. Les conventions suivantes sont adoptées :

- un $ représente le contenu d'un champ alphanumérique non protégé, c'est à dire accessible par l'utilisateur. En pratique, il s'agira donc d'un champ de saisie ;
- un # représente le contenu d'un champ numérique non protégé ;
- un ' représente le contenu d'un champ alphanumérique protégé (ou encore champ d'affichage alphanumérique), accessible seulement par le progiciel, et un ‘ représente le contenu d'un champ numérique protégé.

Tous les autres caractères rencontrés sur la grille sont considérés comme représentant des constantes : lorsqu'ils sont entourés de caractères spéciaux, ils sont entendus comme composant un masque d'édition.

Le dialogue offert par Editor est particulièrement simple et adopte le standard introduit par le traitement de textes Wordstar, pour tout ce qui concerne l'édition proprement dite. Dans le mode d'édition une aide en ligne est disponible pour les débutants.

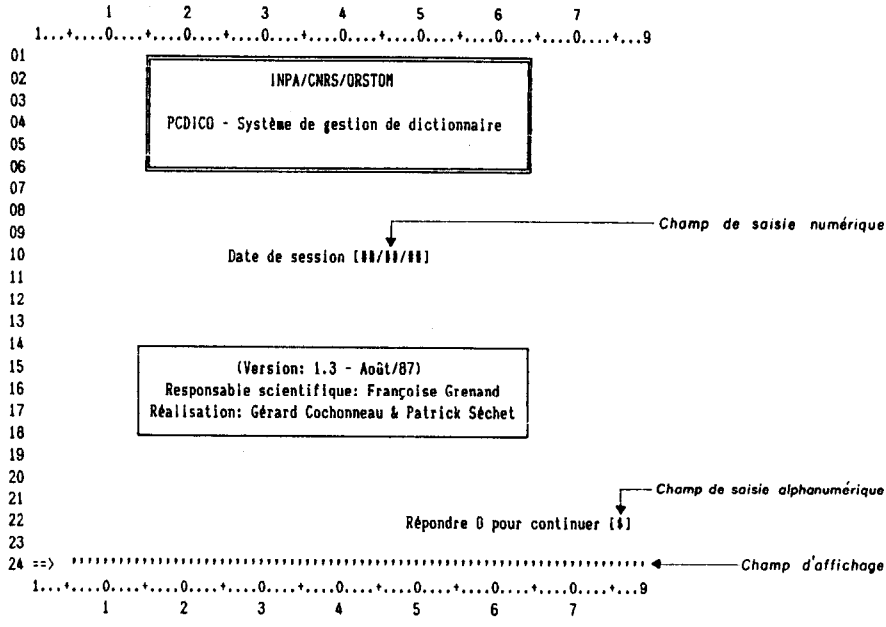
Les fichiers générés par l'éditeur contiennent, en plus de l'image de la grille d'écran, un dictionnaire recevant quelques informations sur les champs variables de l'écran : type, position et masque d'édition.

Un ensemble d'options permet d'administrer les *"fichier-écrans"* : création, mise à jour, sauvegarde, copie, changement de nom et impression. Cette dernière opération génère deux produits :

- la grille d'écran simple, image sur papier de ce qui sera présenté en temps d'exécution sur le moniteur ;
- la grille d'écran complète, qui joint à cette image le listing du dictionnaire des variables contenues sur celle-ci. Le programmeur dispose alors de la liste, du numéro et de la description de chacune de ces variables (Figure 3).

M i c r o - E d i t o r, Copyright, 1986, by Nelio Domingues
Map of: DICDO100.MAP

A. IMAGE DE LA GRILLE D'ÉCRAN



Directory of Variables

B. DICTIONNAIRE DES VARIABLES

Number	Type	Line	Column	Length	!!!	Number	Type	Line	Column	Length
1	#	10	43	6	!!!	2	#	22	76	1
3	'	24	6	74	!!!					

Figure 3 - L'impression sous la forme "écran+dictionnaire" est utile pour le programmeur.

4.2. Telador

Telador se charge de manipuler les fichier-écrans créés par Editor. C'est une bibliothèque (en code source Turbo Pascal : *"book"*) de procédures à inclure dans les programmes applicatifs. Il autorise également la saisie des informations en fournissant certaines facilités élémentaires de manipulation de curseur et édition de champs (déplacement en avant et en arrière d'un caractère, d'un mot, ou d'un champ, insertion d'un caractère, exclusion d'un caractère ou d'un mot, mise à blanc d'un champ, etc.).

Telador permet de réaliser six fonctions : celles-ci peuvent être activées, par le programme-hôte, au moyen de commandes spécifiques qui se présentent sous la forme d'appels de procédures sans paramètre :

- la première, *OpenMap*, ouvre et lit le fichier-écran, rendant ainsi disponibles tous les paramètres des champs de saisie et affichage ;
- une autre routine, *ClearMap*, prépare le contenu des variables, protégées ou non, qui vont recevoir des informations, en changeant les caractères spéciaux ($,#,',') par des espaces ;
- la procédure *WriteMap* réalise la visualisation effective de la grille sur l'écran : cette subroutine est normalement appelée après les deux précédentes, de sorte que la grille apparaît avec les champs variables en blanc. La réunion de ces trois subroutines en une seule aurait amélioré la lisibilité du programme : on a préféré les dissocier pour augmenter les performances dans certaines situations (par exemple, quand une lecture préalable du fichier-écran est souhaitable, la routine OpenMap pourra être appelée seule) ;
- la saisie des informations est réalisée par le biais de la procédure sans paramètre *ReadVar*, quels que soient le numéro et le type des données à récupérer de la grille. Un index est utilisé, et mis à jour à chaque appel de cette routine, pour indiquer quelle variable est reçue par le programme-hôte. Lorsque c'est nécessaire, les commandes de mouvement du curseur ou d'édition de champ s'accompagnent automatiquement de la mise à jour de cet index, de sorte que la saisie est *"full screen"* ;
- la valeur courante de toutes les variables de la grille, qu'elles soient protégées ou accessibles à l'opérateur, peut être écrite sur l'écran par le biais d'une commande spécifique, *WriteVar*, sans qu'il soit nécessaire d'écrire de nouveau toute la grille sur le moniteur. Enfin, la procédure *ClearVar* remet toutes les variables internes du programme-hôte en relation avec le telador à blanc, ce qui permet de traiter facilement le cas fréquent des saisies répétées sur la même grille (plusieurs articles d'un même fichier : mode *"input"*).

Quelques options sont disponibles, comme celle de donner ou non l'écho à l'écran d'un caractère taper au clavier, ce qui permet le traitement adéquat des informations réservées (mot de passe, par exemple), ou encore de transformer systématiquement en majuscules toutes les données saisies.

4.3. Exemple d'utilisation

La séquence de commandes présentée ci-dessous est typique de l'utilisation de Telador dans un programme-hôte, pour la saisie d'informations :

```
OpenMap ;          (* ouvrir et initialiser la grille *)
ClearMap ;         (* mettre à blanc les champs variables *)
WriteMap ;         (* visualiser la grille à l'écran *)
InformerChamps ;   (* valeurs des champs protégés *)
WriteVar ;         (* écriture effective de ces valeurs *)
repeat             (* boucle sur les champs à saisir *)
  ReadVar ;        (* lecture de la variable courante *)
  ValiderChamps    (* critique du champ correspondant *)
until Confirme ;
WriteVar ;         (* écriture de tous les champs *)
```

OpenMap ouvre le fichier-écran qui lui est indiqué par l'intermédiaire d'une variable globale, contenant le nom de ce fichier. La procédure interne InformerChamps, écrite par le programmeur d'applications, est destinée à transmettre certaines données du progiciel à l'utilisateur, comme la date de session, par exemple. Celles-ci seront effectivement montrées à l'écran à la suite de l'appel de WriteVar.

La prise en considération des informations fournies par l'utilisateur final est réalisée à l'intérieur d'une instruction répétitive, ReadVar, qui assure à chaque appel l'incrémentation de l'index pointeur de champ. La procédure interne ValiderChamps, également écrite par l'utilisateur, doit fournir le traitement adéquat à chaque variable : ce pourra être une instruction *"case"* utilisant l'index pointeur de champ comme sélecteur.

Enfin, il est naturel d'incorporer, comme dernière donnée à saisir par l'utilisateur, un champ de confirmation dont le résultat est exprimé sous forme de booléen (Confirme), pour valider définitivement les informations fournies par l'opérateur. A l'issue de la saisie, un nouvel appel à WriteVar exhibe l'ensemble des valeurs des variables, telles qu'elles ont été enregistrées par le progiciel.

5. AUTRES GESTIONNAIRES D'ECRAN

L'utilisation de ce premier outil, par l'équipe franco-brésilienne du programme SISGEO lors de la mise au point de deux applications différentes, a confirmé tous les bénéfices que l'on pouvait en tirer, et qui ont été mis en évidence au début de cet article.

L'assimilation de cette technique par les programmeurs peut être considérée comme satisfaisante, bien que dans la plupart des cas ceux-ci tardent à percevoir tout le potentiel d'un tel outil. L'adoption d'un gestionnaire d'écrans plus complet et plus sophistiqué a donc été décidée. Celle-ci coïncide avec l'exclusion des micro-ordinateurs de 8 bits, sous CP/M, comme configuration minimale d'opération des modules locaux développés dans le cadre du projet.

Une étude rapide des progiciels disponibles susceptibles d'assurer la création, l'administration et l'utilisation des écrans et fenêtres dans l'environnement PC-DOS a donc été menée : ses résultats sont présentés ci-après.

5.1. Telador, version 2.0

Dans un premier temps on a cherché à mettre au point une nouvelle version de *"Telador"*, essentiellement dans l'optique d'éliminer les inconvénients majeurs qui avaient été détectés lors de l'utilisation de la première version. Ces inconvénients sont les suivants :

- la syntaxe adoptée pour les fonctions utilisant les écrans avait été choisie pour apporter un maximum de souplesse : en contrepartie, les programmes s'en trouvent parfois un peu alourdis (une affectation et trois appels de procédure pour montrer un écran, par exemple) ;
- la taille des écrans est relativement importante, compte tenu de la présence du dictionnaire associé à la grille. La taille minimale est de 2K (25 lignes x 80 colonnes) : elle croît avec le nombre (et la taille) des variables contenues dans la grille, pouvant atteindre 4K. cela est surtout gênant dans l'environnement PC-DOS, quand l'espace disponible sur une unité de volume standard (une disquette) est de 360K ;
- le nombre des fichiers créés devient relativement important, dans la mesure où chaque grille d'écran constitue un fichier. Bien que le nombre des écrans créés varie surtout avec le *"confort"* que l'on prétend fournir à l'utilisateur, les deux applications qui ont été mises au point peuvent servir d'exemple, la première utilisant dix écrans (6.000 lignes de code source), la seconde près de soixante (45.000 lignes de code source). On remarquera que ce problème est cette fois plus sensible dans

l'environnement CP/M, quand le nombre maximal de fichiers par volume est nettement plus limité (64). De plus, un nombre de fichiers élevé allonge le temps d'installation de l'application et tend à confondre l'utilisateur final ;

- certaines opérations, comme la visualisation d'un écran ou l'affichage des champs sont relativement lentes. Le fait qu'elles soient écrites en langage de haut niveau est une des raisons de cette lenteur. De plus, comme on a cherché à éliminer au maximum les paramètres des procédures, l'appel à WriteVar écrit systématiquement tous les champs sur l'écran, ce qui est extrêmement peu performant.

Telador 2 simplifie le plus possible l'interface avec le programme hôte : une procédure unique est utilisée et est appelée avec trois paramètres : nom de l'écran, numéro de la variable à considérer et nom de zone de transfert. L'affichage d'une variable est ainsi réalisé en fournissant le champ à visualiser dans la zone de transfert : si la grille n'est pas présente à l'écran, elle est automatiquement visualisée. Le résultat d'une saisie est également obtenu dans cette même zone.

Pour réduire l'encombrement du fichier-écran, on a exploité le fait que la grille contient, en elle-même, toutes les caractéristiques des champs (format, position, longueur et masque sont fournis par les caractères spéciaux utilisés). Telador 2 utilise donc des fichiers de taille fixe (2K) et construit le dictionnaire de variables chaque fois qu'il est nécessaire.

Le problème de la multiplication des fichiers, causé par le nombre important d'écrans associés à une application, ne pouvait être contourné qu'en regroupant ceux-ci dans des bibliothèques. Telador 2 tire profit de ce que ses écrans sont de longueur fixe pour administrer efficacement des bibliothèques d'écrans constituées par des fichiers séquentiels indexés, offrant toutes les facilités de gestion (création de bibliothèque, inventaire, accès, modification, inclusion et exclusion de grilles d'écran).

Telador 2 offre naturellement une meilleure performance que la première version, ne serait-ce que parce qu'il utilise l'allocation dynamique. De plus l'affichage isolé d'un champ est autorisé, ce qui élimine le problème présenté par la procédure WriteVar. Enfin, on a pas voulu pour cette seconde version se préoccuper de la portabilité vers le micro de 8 bits, ce qui a permis d'utiliser la visualisation par adressage direct en mémoire, réalisée par appel au système d'exploitation DOS.

5.2. Masque

Il s'agit d'un générateur-gestionnaire de grilles d'écran conçu et mis au point par Gérard Cochonneau, de l'ORSTOM. Le module d'utilisation des écrans étant écrit en assembleur 8088, il est utilisable dans plusieurs environnements de programmation (Basic compilé, Fortran-77, Cobol et Turbo-Pascal), ce qui en constitue la caractéristique la plus intéressante.

L'éditeur de grilles d'écran fourni avec Masque s'appuie sur un procédé un peu différent de celle des autres produits présentés dans cet article. En effet, la grille est vue comme un ensemble de lignes, chacune d'elles pouvant contenir un ou plusieurs champs : de cette façon, l'éditeur n'offre et n'utilise aucune commande *"plein écran"*. Dans ces conditions, plutôt que sur la grille d'écran proprement dite, c'est sur une fenêtre de définition de champ que l'analyste travaille pour dessiner et définir sa grille d'écran.

Un champ peut être constant ou variable. Pour définir une ligne d'écran ou n'apparaît ni champ de saisie ni d'affichage, on choisit le type fixe et on éditera alors la ligne entière : l'éditeur ignore la notion de zone ou de bloc. Pour définir un champ variable on utilise la même fenêtre en fournissant explicitement la position désirée (ligne et colonne du premier caractère à occuper par le champ). La ligne en question est alors automatiquement visualisée dans la fenêtre.

A chaque champ variable est associé un nom de champ, qui est utilisé par le programme d'application pour communiquer avec la grille : cette notion de nom de champ est très intéressante, dans la mesure où elle favorise grandement la lisibilité du programme (les fonctions de saisie et affichage devant invoquer explicitement le champ par son nom), en plus d'apporter une relative indépendance entre le programme et la grille. On regrettera toutefois que Masque ne réserve que trois positions à cet effet.

Le logiciel ne comporte pas la notion de type de champ (numérique, alphanumérique, etc.), ceci parce que les langages ne représentent pas tous les types de la même façon. Astucieusement, le type de champ est construit caractère par caractère, chaque position d'un champ ayant elle-même son propre type, choisi parmi A pour alphabétique, X pour alphanumérique, D pour numérique, S pour signe et E pour ASCII étendu (tous caractères imprimables). Cette ressource autorise la définition de champs mixtes (partie numérique et partie alphabétique, par exemple).

L'éditeur permet également d'associer un certain nombre d'attributs à chaque champ : masque d'édition, champ obligatoire ou non, champ à remplir totalement, valeurs initiale, maximale et minimale, ces dernières valables quels que soient les types des caractères qui composent le champ. Il est intéressant de noter ici que Masque ne fait pas de différence entre champ de saisie et

champ d'affichage (cette distinction ne présente en fait un intérêt qu'en mode de saisie pleine page, quand les champs d'affichage doivent être protégés d'une saisie intempestive).

L'éditeur permet bien entendu de visualiser la grille-écran telle qu'elle se présente à l'utilisateur final en temps d'exécution, et même de la tester, en simulant une utilisation réelle de la grille, pour vérifier l'adéquation correcte des attributs qui ont été définis pour les champs variables.

L'impression de l'écran est possible et fournit, avec l'image de l'écran, un tableau des caractéristiques de chaque champ variable contenu dans la grille. Enfin, le logiciel permet la conversion des écrans générés par un éditeur de texte quelconque, moyennant une opération dite de compilation et quelques restrictions concernant le format des fichiers d'entrée.

L'utilisation, dans un programme-hôte, des fichier-grilles générés par cet éditeur nécessite l'incorporation de Masque, comme module externe (appelé au moment de l'édition de liens, quand elle existe, ou déclaré *"external"*). La communication avec le module est réalisée par le biais des paramètres effectifs au moment de l'appel, en nombre variable selon la fonction désirée. La gestion de ce type d'interface variant d'un langage à l'autre, on se bornera à décrire l'utilisation de Masque dans un programme Turbo Pascal.

Sept fonctions au total sont autorisées : le code de la fonction à exécuter est passée au module à l'aide d'une variable globale (Op), sous forme d'une chaîne de deux caractères (par exemple RD pour Read, WR pour Write). L'appel de la procédure (Mask) doit présenter systématiquement deux paramètres, pour satisfaire les exigences de cohérence chères au langage Pascal. Toutefois, suivant la fonction appelée, l'un d'eux, et parfois les deux seront inutiles (*"dummy"*).

La séquence d'instructions présentées ci-après est typique de l'utilisation de Masque dans un programme d'application :

```
Op := 'CL' ;                (* appel de la fonction Clear *)
Mask(Dummy,Dummy) ;         (* pour effacer l'écran *)
...
Grille := 'DEMO.' ;
Op := 'GT' ;                (* appel de la fonction Get *)
Mask(Grille,Dummy) ;        (* pour afficher l'écran *)
...
Nom := 'DAT.' ;
Valeur := '12/08/87' ;
Op := 'WR' ;                (* appel de la fonction Write *)
Mask(Nom,Valeur) ;          (* pour afficher un champ *)
...
```

```
Nom := 'TEL.' ;
Op := 'RD' ;                    (* appel de la fonction Read *)
Mask(Nom,Valeur) ;              (* pour récupérer un champ *)
if Stat <> '00'
  then Erreur ;                 (* test du code de retour *)
```

La fonction Get doit être utilisée pour afficher l'écran dans son état initial. Dans cet exemple, on utilise la fonction Write pour afficher le contenu de la variable *"valeur"* dans le champ indiqué par *"nom"*. De la même façon, la fonction Read autorise l'opérateur à saisir le champ nom, et transmet le résultat au programme-hôte par le biais de la variable valeur. A l'issue de cette opération, la variable globale Stat, qui constitue le code de retour des fonctions de Masque, peut être testée pour vérifier le résultat de la fonction. En particulier, si une touche de fonction a été actionnée, Masque retourne le code correspondant au programme-hôte dans cette variable, ce qui permet de prévoir certaines actions (écran d'aide, abandon de la saisie, etc.).

Tant pour la fonction Read que pour la fonction Write, le logiciel permet de fournir plusieurs noms de champs concaténés dans le premier argument, ce qui autorise un affichage efficace des champs de l'écran, d'une part, la saisie en mode pleine page, d'autre part.

Trois autres fonctions sont fournies :

- Reset, pour remettre la grille d'écran dans son état initial (ou seulement un ou plusieurs champs variables), sans avoir à relire le fichier ;
- Display, pour visualiser un message d'information dans la vingt-cinquième ligne de l'écran ;
- Output Color, pour afficher une chaîne de caractères à partir d'une position quelconque de l'écran, avec le choix des attributs vidéo (et/ou de la couleur).

Les trois logiciels présentés ci-après sont des produits du commerce et, comme tels, fournissent des prestations plus complètes, essentiellement dans la partie d'édition. Ils permettront normalement de créer plus *"confortablement"* les écrans, ces derniers étant également plus soignés.

La description succincte qui en est donnée ci-après poursuit un objectif d'analyse comparative basée sur les fonctions d'un gestionnaire-générateur d'écrans : l'éditeur, en distinguant l'édition des zones constantes (le décor), la définition des champs (et les caractéristiques disponibles) et la gestion des grilles, d'une part, la bibliothèque d'utilisation, sa forme, sa syntaxe et ses ressources, d'autre part. Les fonctions d'édition de champ disponibles en temps d'exécution constituent un dernier critère, particulièrement important pour l'utilisateur final.

5.3. Display Manager

Ce gestionnaire d'écrans destiné à l'IBM-PC est un produit commercialisé depuis 1983 par Digital Research. Il a la particularité d'offrir, en plus des deux parties classiques d'édition et d'utilisation de grilles d'écran, un programme d'installation qui autorise le paramétrage des codes particuliers à chaque moniteur (taille de l'écran, positionnement du curseur, attributs vidéo, etc.).

L'éditeur est conçu pour générer des écrans numérotés, lesquels sont stockés dans un fichier appelé *"Display File"*. Plusieurs de ces bibliothèques peuvent être créées pour une application déterminée, chacune pouvant compter jusqu'à 250 écrans. L'éditeur distingue la partie constante des écrans (champs littéraux) de la partie variable, classée en champs d'entrée (de saisie) et champs de sortie (affichage).

Alors que la partie constante de l'écran ne peut recevoir aucun attribut vidéo sur un moniteur monochrome (surbrillance, vidéo inversée, soulignement, clignotement), ni couleur sur un moniteur polychromatique, les champs de saisie et d'affichage reçoivent un numéro de référence et l'éditeur permet de faire varier leurs attributs vidéo (resp. couleurs), pour en accroître le contraste et la visibilité. En fonction des caractéristiques du moniteur utilisé au moment de l'exécution, les attributs vidéo ou les couleurs définies pour le champ seront activés, ce qui évite le risque de mauvaise lisibilité lié à un choix unique.

Les ressources fournies par l'éditeur pour dessiner les écrans suivent sensiblement le standard Wordstar (mouvements du curseur, effacement, etc.). Il n'y a pas de commandes sur les blocs, mais des commandes spéciales d'édition de champ, préfixées <Ctrl>U, la position de ceux-ci devant être marquée sur la grille d'écran.

Pour définir toutes les caractéristiques des champs variables, Display Manager utilise le concept de *"Status Window"*, associant à chaque champ une fenêtre de paramètres qui doivent être fixés par l'analyste de spécification. En plus des attributs vidéo et couleurs, le choix d'un format spécifiant comment le contenu du champ doit être présenté dans l'espace qui lui est réservé (justification à gauche ou à droite, position du point décimal, code monétaire, etc.) est possible.

Dans le cas particulier des champs de saisie le choix d'un type entre alphabétique, alphanumérique (avec ou sans caractères spéciaux, avec ou sans écho, avec ou sans conversion en majuscules), entier ou réel est autorisé. L'information donnée par l'utilisateur en temps d'exécution est alors validée, caractère par caractère, en fonction de ce type. Le résultat provoqué par la saisie

d'un caractère interdit est également contrôlable par la *"status window"* de chaque champ (*beep*, fin de champ automatique), ainsi que le traitement (fin de champ ou caractère illégal) qui doit être réservé aux flèches de mouvement du curseur et aux touches de fonction.

Display Manager permet l'impression de chaque grille-d'écran avec le dictionnaire associé des champs d'entrée et sortie : cette impression peut être réorientée dans un fichier disque, le logiciel permettant la concaténation de celles-ci dans l'optique de faciliter la réalisation du manuel d'utilisation pour l'application en cours de mise au point.

Pour l'utilisation des grilles d'écran générées par l'éditeur, on dispose d'un certain nombre de fonctions. Le programmeur d'application doit alors en codifier les appels, le code propre aux fonctions étant inclus dans la phase d'édition de liens, à partir de la bibliothèque objet fournie par Display Manager. Ces fonctions doivent évidemment être déclarées comme externes, ce qui oblige à incorporer les déclarations correspondantes (fournies) dans le programme source.

Ce mode de fonctionnement interdit l'utilisation de ce produit par les versions interprétées (ou compilées mais sans étape d'édition de liens) des langages. De plus, la codification par fonctions (et non par procédures) en empêche l'utilisation en Cobol, langage qui n'accepte pas cette notion. Display Manager est néanmoins utilisable en Basic (CBasic), C (Digital Research), Pascal (Pascal MT+ 86) et PL1 (PLI-86).

Les fonctions de Display Manager retournent normalement une variable entière (code de retour), dont la valeur est zéro si l'opération s'est bien passée, une valeur négative dans le cas contraire. La séquence d'instructions ci-dessous est typique de l'utilisation de ces fonctions dans un programme d'application écrit en Pascal :

```
var
  Retour  : integer ;
  Saisie  : string[80] ;
...
Retour  := OpnDis('ECRANS.DIS') ;     (* ouvrir display file *)
Retour  := DispD(1) ;                 (* montrer l'écran n 1 *)
Retour  := PosF(3) ;                  (* choisir le champ 3  *)
Retour  := PutF('Titre') ;            (* affichage du champ  *)
Retour  := NxtF(-2) ;                 (* positionner         *)
Saisie  := GetF ;                     (* récupérer l'entrée  *)
```

L'appel à la fonction InitDM, dont le rôle est de fixer les caractéristiques du moniteur utilisé à l'exécution, doit être fait avant toute utilisation des autres fonctions. Une bibliothèque doit ensuite être ouverte et un écran particulier chargé : il n'est pas possible d'avoir plus d'un écran actif à la fois, ce qui est un facteur limitant pour l'utilisation de fenêtres dans une application.

L'affichage d'un texte dans un champ est assuré par la fonction PutF, tandis que le contenu d'un champ de saisie est récupéré (et validé) par l'intermédiaire de la fonction GetF, toutes deux agissant sur le champ courant. Le positionnement absolu est obtenu par PosF, le positionnement relatif par NxtF, dont l'argument permet plusieurs variations (champ suivant, champ précédent, d'entrée ou de sortie, dernier ou premier champ).

Une valeur défaut peut être donnée (par PutF) pour un champ de saisie, constituant de cette façon un guide pour l'utilisateur : la fonction UpdF devra alors être utilisée à la place de GetF, pour récupérer le contenu du champ même si l'utilisateur n'a entré aucun nouveau caractère.

Par ailleurs, Display Manager fournit quelques ressources sophistiquées pour modifier les attributs d'un champ ou tester les capacités du moniteur en temps d'exécution.

5.4. Screen Maker

Ce produit commercialisé depuis janvier 1987 dans sa version 1.3 par *"Logiciel et Médias"* est un éditeur d'écrans qui génère une procédure d'affichage en code source (Turbo Pascal), auquel est associé un ensemble de librairies complémentaires autonomes, également en code source, facilitant l'utilisation des écrans et fenêtres.

Screen Maker s'accompagne d'un éditeur qui permet de créer ou modifier un écran, ou une fenêtre, à la manière d'un éditeur de texte standard, mais en proposant des fonctions supplémentaires de définition des champs d'entrée/sortie ou de génération et gestion de fenêtres. Les commandes de cet éditeur font un plein emploi des ressources du clavier (flèches, touches de fonction, etc.) pour produire les mouvements du curseur, l'effacement, le centrage, les commandes sur les blocs, le passage en mode insertion, etc..

Un mode spécial de modification de bloc permet de traiter globalement des parties d'écran, appelées blocs, permettant de copier, déplacer, d'effacer un bloc, en modifier et en contrôler la taille, etc.. Le bloc sert également de cible aux commandes appelées de *"boîtes et lignes"*, destinées à tracer des cadres et des lignes de séparation dans un bloc. Couleurs et attributs vidéo sont également gérés de façon très complète. Le mode dessin est une caractéristique intéressante de l'éditeur fourni par Screen Maker : il permet

de tracer une figure de forme quelconque, simplement en agissant sur les touches de direction du curseur.

La définition des champs d'entrée/sorties est réalisée durant la phase d'édition de l'écran : ils sont repérés par des caractères spéciaux, différents pour les champs de saisie et d'affichage, placés dans les positions désirées sur l'écran. Le type de champ, spécifié par une lettre située après le dernier caractère du champ à caractériser, est choisi entre *s pour string, i pour entier, r pour réel, t pour heure et d pour date.* Ces deux derniers champs, s'ils existent, provoqueront l'affichage de l'heure et de la date système aux endroits indiqués.

Les attributs vidéo et la couleur des champs peuvent être définis indépendamment, champ par champ : par contre, certaines autres caractéristiques (champ obligatoire ou non, écho ou non) devront être gérées par le programme-hôte.

Screen Maker associe l'impression de l'écran, et du dictionnaire de variables, à l'opération qui consiste à générer la procédure en code source destinée à l'administration de cet écran. Le choix du label de cette procédure est laissé à critère de l'utilisateur, ainsi que le nom du fichier qui la contiendra et qui devra être inclus dans le programme d'application.

Pour l'utilisation des écrans et fenêtres générés par l'éditeur, il est nécessaire d'inclure certaines bibliothèques fournies par le logiciel, en fonction des ressources utilisées :

- Video1.usr, doit toujours être inclue ;
- Windiws1.usr, correspond à l'affichage des fenêtres ;
- Keyboard.usr, Datetime.usr et Scrdrive.usr sont nécessaires pour pouvoir gérer les champs d'entrée/sortie.

Screen Maker fournit une procédure spéciale, EditFields, pour traiter un écran globalement en éditant les champs d'entrée en mode pleine page. Pour une édition champ par champ on aura la séquence d'instructions typique suivante :

```
(*$I SCREEN.COD *)              (* fichier de code généré *)
     ...
     InitVideo ;              (* initialise l'adresse écran *)
     TabScreen ;              (* initialise les variables *)
     Screen(0) ;              (* visualisation de l'écran *)
     EditFields ;
     PutField(var,1,p1,p2) ;  (* affiche le champ 1 *)
     GetField(var,2) ;        (* saisie du champ 2 *)
```

L'appel Screen(0) visualise l'écran SCREEN dans la page 0 : le logiciel permet l'utilisation des quatre pages de mémoire d'écran disponibles sur la carte graphique en mode 80 colonnes (procédure DisplayPage), d'où la possibilité d'un affichage instantané des

écrans contenus dans des pages différentes, puisque cette opération correspond à un simple changement de l'adresse de base du contrôleur d'écran.

La procédure PutField affiche le contenu de la variable (sans type) var dans le champ indiqué par l'entier qui constitue le deuxième paramètre de l'appel. Les deux paramètres suivants, sont des booléens qui indiquent certaines caractéristiques qui varient suivant le type de champ (centrage, passage en majuscules, signe d'un champ numérique, etc.).

La procédure GetField récupère le contenu du champ d'entrée indiqué comme second paramètre de l'appel, dans une variable sans type (var). L'appel des procédures standards InitVideo et TabScreen est nécessaire pour initialiser les opérations et le tableau des variables de l'écran.

Un certain nombre de fonctions complémentaires sont disponibles pour la gestion des champs : elles s'expriment sous forme de procédures contenues dans la bibliothèque Scrdrive.usr, appelables par le programme-hôte. De cette façon, on peut imposer des limites minimale et maximale aux champs de saisie numériques (SetLowerLimit et SetUpperLimit), imposer un masque de saisie pour l'entrée de chaînes de caractères (SetMaskTo), initialiser un champ de saisie avec une valeur défaut (InitInputField), manipuler les erreurs de saisie dans le programme d'application (UserErrorRoutine), etc..

On remarquera que cette manière de définir les attributs des champs transfère le travail de validation correspondant à la phase de programmation (il ne sera d'ailleurs pas nécessairement plus simple que s'il avait été codifié directement avec les instructions de base du langage), alors que quand ces mêmes caractéristiques peuvent être définies au moment de l'édition des écrans, le travail correspondant est réalisé en temps de spécification, ce qui paraît plus adéquat.

La bibliothèque Datetime.usr fournit les routines nécessaires à la mise à jour des champs correspondants à partir de la date et de l'heure du système.

5.5. Turbo Screen

Il s'agit d'un progiciel distribué par la société PC-Soft : la version 1.2 est datée de février 1987. Egalement destiné à la génération d'écrans et fenêtres, et à leur utilisation dans un programme d'application codé en Turbo Pascal, ce produit se distingue du précédent par le fait que la bibliothèque contenant les procédures de gestion des écrans constitue un module résident unique, Affturbo.com, qui doit être chargé en mémoire préalablement à l'exécution.

Cette solution offre un avantage déterminant sur la précédente, dans la mesure où le segment de code réservé au programme est entièrement disponible, alors que dans le cas de l'inclusion des routines en code source, cet espace (64K) se trouve sensiblement réduit (de 12K environ pour Telador, jusqu'à 26K pour Screen Maker en utilisant toutes les options disponibles dans ce dernier logiciel). La souplesse apportée en contrepartie, par l'inclusion de procédures en code source (modifiables par le programmeur), n'est normalement pas relevant, d'autant plus que toute modification sera extrêmement préjudiciable à la maintenance ultérieure des programmes.

L'éditeur d'écrans disponible avec Turbo Screen est particulièrement complet et offre l'essentiel des fonctionnalités que l'on peut attendre d'un éditeur moderne (aide en ligne, gestion des caractères semi-graphiques, tracé de cadres, coloriage de zones, paramétrage des touches de fonction, manipulation de fichiers et changement de *"drive"* ou de répertoire, etc.). Il est possible à tout moment de faire appel au système d'exploitation et de revenir à l'éditeur, sans incidence sur le travail en cours. Il offre la particularité de sauvegarder écrans ou fenêtres sous la forme de fichiers, ou sous la forme de bibliothèques (fichiers séquentiels, d'où une gestion plus rudimentaire, mais simplifiée).

La partie constante des écrans est traitée sous la forme de zones et il n'y a pas de distinction à priori entre les champs de saisie et ceux d'affichage. La notion de zone permet une manipulation commode avec les instructions disponibles de définition, déplacement, recopie, effacement, coloriage, tracé de cadre : ces zones peuvent de plus être sauvegardées isolément (fenêtres).

Pour la génération d'écrans *"portables"* couleur/monochrome, l'éditeur offre la possibilité de visualiser l'image-écran sur deux moniteurs connectés simultanément, ce qui permet de s'assurer de la lisibilité sur les deux types d'écran.

Les champs sont repérés par leur position sur la grille et se voient attribuer un nom de champ, qui sera utilisé par les commandes de saisie et affichage dans les programmes applicatifs. La définition des caractéristiques d'un champ se fait dans une fenêtre analogue à la *"Status Window"* de Display Manager, avec sensiblement les mêmes ressources : validation automatique ou non, champ obligatoire, saisie invisible, transformation en majuscules, divers types de validation et cadrages, couleurs, etc..

Dans le cas de champs numériques, entier ou réel, valeurs minimale et maximale peuvent être stipulées, la valeur entrée étant automatiquement contrôlée en fin de saisie. La cohérence d'un champ de type date (nombre de jours du mois, etc.) est également testée.

Enfin, ce progiciel fournit deux notions particulières : tableau de champs et message associé à un champ de saisie. Le tableau de champs est très utile dans certaines situations de capitalisation d'information, quand il simplifiera grandement la codification. Par ailleurs, un message peut être associé à chaque champ : il s'affiche automatiquement en 25ème ligne lors de la saisie, et s'efface dès validation (avec restauration du contenu antérieur de la ligne).

Turbo Screen permet l'impression de chaque grille-écran avec ou sans le dictionnaire : une impression particulière, de type *"hard copy"*, reflète les effets spéciaux (attributs vidéo).

L'utilisation des grilles d'écran générées par ce progiciel est assurée par le module résident Affturbo.com, qui intercepte les ordres passés par le programme. Cette communication est réalisée à partir des fonctions classiques d'affichage, Write, et de lecture au clavier, ReadLn, identifiées par des caractères spéciaux (délimiteurs de début et fin de commande).

Le nom de la fonction à exécuter, d'une part, les paramètres de cette fonction, d'autre part, sont alors donnés sous forme d'une chaîne de caractères. Certaines de ces fonctions retournent une valeur (ou compte-rendu, CR), nulle quand tout s'est bien passé et qui peut être testée par le programme-hôte.

Turbo Screen autorise le traitement des fenêtres et écrans selon deux modes distincts : le mode champ à champ et le mode pleine page. Dans le premier mode le programme-hôte gère lui-même le parcours de champ à champ, en effectuant n'importe quel traitement après la validation de chaque champ (décodification et affichage d'un libellé à partir d'une table, ouverture d'une fenêtre, etc.). En mode pleine page, c'est le module résident qui gère seul les déplacements de champ à champ et le programme-hôte ne *"reprend la main"* qu'après validation de la totalité de la fenêtre. Cette dernière solution est moins souple, mais réduit considérablement le code source nécessaire : elle est bien adaptée au cas de saisie de fiches entières.

Le mode pleine page dispose d'un jeu de fonctions spéciales qui permettent la sélection des champs à saisir et l'ordre dans lequel ils doivent être saisis.

La séquence de commandes ci-après, qui réalise la visualisation d'un écran, l'affichage d'un champ (TVA), la saisie d'une variable (Prix) et son passage au programme principal, est typique de l'utilisation d'écrans dans ce progiciel, en mode champ par champ :

```
const
  fin = chr(01) ;
  ...
  Write(deb,'UTILISE,ECRAN1.AID',fin) ;     (* montrer écran1 *)
  Write(deb,'AFFICHE,TVA,=18.60',fin) ;     (* affichage      *)
  Write(deb,'SAISIE,Prix',fin) ;            (* saisie         *)
  ReadLn(Prix) ;                            (* récupération   *)
```

Alors que le délimiteur de fin est une constante, celui de début est une fonction qui permet d'adresser la recherche de la routine de Turbo Screen à exécuter dans le segment de mémoire contenant le module résident Affturbo.com.

Un grand nombre d'options sont disponibles, comme le *"scrolling"* de l'écran (fonctions MONTE et DESCEND), l'affichage de messages (fonctions MESSAGE et ERREUR), l'utilisation de la bibliothèque d'écrans (fonctions BIBLI et MEMO), la gestion des fenêtres en recouvrement (fonctions OUVRE, FERME et FUSIONNE), le stockage et la récupération de l'écran en cours (fonctions SAUVE et RETOUR), la définition de l'action des touches spéciales dans le contexte de saisie (SORTIE), le changement des couleurs en temps d'exécution, la gestion de menus, classiques ou déroulants, en tant qu'écrans spéciaux.

En temps d'exécution l'utilisateur final dispose pour la saisie des ressources d'édition standard, éventuellement modifiées par la fonction SORTIE : <Home>, <End>, -->, <--, <Del>, <Ins>, <Esc> et <CR> ont une action conforme à l'expectative. En outre, si le mode pleine page a été défini, l'utilisateur pourra se déplacer à son gré à l'aide des quatre touches fléchées et de la touche <Tab>.

Enfin, Turbo Screen fournit trois ressources complémentaires intéressantes :

- un mode test (type *"Trace On/Off"*) qui permet d'afficher sur la 25ème ligne chaque fonction demandée au module résident, dans le but de faciliter la dépuration du programme ;
- la possibilité de fusionner deux écrans : celle-ci peut être utilisée pour introduire des caractères de grand format, correspondants à des polices fournies par le progiciel ;
- la possibilité d'effectuer un traitement parallèle (affichage de l'heure, par exemple) pendant la saisie des champs, moyennant l'utilisation d'une variable spéciale de compte-rendu, CR+, qui rend la main au programme-hôte tant que la fonction de saisie n'a pas été validée.

CONCLUSION

L'examen de plusieurs produits différents, pour la génération et l'utilisation de grilles d'écrans sur micro-ordinateurs PC-compatibles et dans des environnements de programmation généralement utilisés sur ce type de matériel, met en évidence un certain nombre de caractéristiques intéressantes. Celles-ci sont plus ou moins développées selon les produits et pourraient devenir des critères de choix entre ces logiciels.

Parmi les caractéristiques qui concernent l'éditeur proprement dit, on appréciera particulièrement la possibilité de création de fenêtres : les produits qui offrent la notion de zone (et cadres), comme Turbo Screen et Screen Maker, y sont bien préparés. L'édition de la grille telle qu'elle se présente à l'utilisateur final est une ressource minimale que tous les produits offrent.

La réalisation d'écrans très soignés et d'apparence professionnelle exige la maîtrise complète des attributs vidéo et de la couleur (Display Manager et surtout Turbo Screen et Screen Maker). Dans cet ordre d'idées, Display Manager est le seul produit permettant un paramétrage complet des ressources des moniteurs utilisés en temps d'exécution et, partant, autorise une visualisation optimale.

Les options disponibles à l'impression de la grille d'écran (avec ou sans le dictionnaire des champs variables) constituent un outil précieux pour la réalisation de la documentation du système applicatif mis au point.

La définition des champs à l'aide d'une fenêtre spécifique (Masque, Turbo Screen et Display Manager) est sans doute la solution la plus adéquate pour permettre de stipuler toutes les caractéristiques des champs de saisie et affichage. Celles-ci sont importantes dans la mesure où, lorsque les valeurs possibles des champs sont totalement spécifiées, la validation est prise en compte par le gestionnaire d'écrans, ce qui décharge d'autant la codification du programme principal.

Il est donc intéressant d'avoir la possibilité de préciser un maximum de paramètres, tels que format de champ, champ obligatoire ou non, valeurs défaut, minimale et maximale, etc.. Un traitement à effectuer sur un champ (cadrage, transformation automatique en majuscule, saisie invisible) peut également être défini, de façon à économiser du code dans le programme d'application : pour ce type de ressource, Turbo Screen est le plus complet, suivi de près par Display Manager et Masque.

En ce qui concerne l'utilisation des écrans, la notion de mode de saisie (pleine page ou champ par champ) est importante : elle est pourtant peu commentée par tous ces progiciels. Seul Turbo Screen met en oeuvre explicitement ces deux modes d'entrée, en

fournissant une procédure spécifique (Ecran) pour le mode pleine page.

A ce propos, la distinction entre champ de saisie et champ d'affichage, placée au premier plan par plusieurs produits (Editor/Telador, Display Manager et Screen Maker), n'apparaît guère importante qu'en mode de saisie pleine page. Turbo Screen et Masque dispensent facilement cette distinction, en autorisant la sélection des champs à saisir (et l'ordre de saisie) : transférer cette tâche au programme d'application ne paraît pas pénalisant outre mesure.

La plupart de ces logiciels ne permettent pas l'ouverture simultanée de plusieurs grilles (exception faite de Screen Maker) : en première analyse ceci devrait constituer un obstacle pour l'utilisation des fenêtres, ou du moins un facteur limitant (fenêtres relativement simples, sans champ de saisie).

La syntaxe des fonctions des gestionnaires d'écran est un sujet de préoccupation, du moins pour les utilisateurs (programmeurs d'application), dans la mesure où il ne semble pas en avoir été un pour les concepteurs. L'appel d'une procédure unique (Masque et Telador 2) pourrait être une solution : mais lorsque beaucoup de fonctions sont proposées, le problème des paramètres devient vite délicat et on confond rapidement ceux à fournir de ceux obtenus. Pour des raisons évidentes de lisibilité, les appels de procédure sont préférables à des appels de fonction (Display Manager). Dans ce domaine, un effort certain a été fait par Turbo Screen, avec des vocables tout à fait expressifs en français (certains le regretterons), comme UTILISE, AFFICHE, DESCEND, etc. : dommage que cet effort soit partiellement gâché par l'utilisation dans une chaîne de caractères, entre deux délimiteurs et à l'intérieur d'une procédure d'entrée/sortie standard de Turbo Pascal !

Le repérage des champs variables par des noms (Turbo Screen et Masque), plutôt que par des numéros, est de nature à améliorer substantiellement la lisibilité des programmes d'application.

L'espace occupé, en temps d'exécution, par le gestionnaire d'écrans constitue un facteur tout à fait relevant, principalement lorsque l'on ne dispose que d'un seul segment pour le programme. Cet espace croît bien entendu avec la panoplie de fonctionnalités offertes par cette partie du logiciel et, selon les versions, varie entre 12 et 26K. La solution du module résident, apportée par Turbo Screen, constitue un élément décisif pour les utilisateurs de Turbo Pascal, puisqu'il laisse intact le segment de code.

Le problème de l'espace occupé par les fichier-écrans n'est normalement pas aussi crucial : selon le produit, et le nombre de variables contenues sur la grille, la taille du fichier peut varier de 2

à près de 10K : seul Telador 2, qui ne stocke que l'image de l'écran, produit des écrans de taille fixe, et minimale, 2K. Pour cette raison, il offre une manipulation complète de bibliothèque d'écrans : Display Manager offre sensiblement la même capacité avec le *"Display File"*, et Turbo Screen permet d'opter pour grilles isolées ou bibliothèque, ce qui constitue une particularité intéressante.

En ce qui concerne les ressources en temps d'exécution, les produits spécifiques à l'IBM-PC utilisent normalement les touches fléchées : seul Editor/Telador, opérationnel sur micro 8 bits, conserve le standard Wordstar. On notera de plus que Display Manager, Turbo Screen et Masque permettent le paramétrage des touches spéciales du clavier de l'IBM-PC, en définissant leur action en saisie (abandon, validation, champ suivant, champ précédent, etc.). Cette caractéristique permet d'augmenter la convivialité des applications mises au point.

L'utilisation d'un même générateur/gestionnaire de grilles d'écrans avec plusieurs langages de programmation différents n'est le privilège que de Masque et Display Manager, le premier couvrant pratiquement toute la gamme des langages les plus utilisés à l'heure actuelle sur les PC-compatibles. Cet aspect pourra constituer un facteur décisif de choix dans certaines situations, où plus d'un langage est utilisé dans une même application, parfois seulement pour des raisons historiques.

REFERENCES BIBLIOGRAPHIQUES

Digital Research : *Display Manager, productivity tool user's guide for the IBM Personal Computer Disk Operating System*, Pacific Grove, USA, july 1983.

PC/SOFT Informatique : *Manuel de référence Turbo Screen, pour les ordinateurs personnels IBM et compatibles*, Montpellier, février 1987.

P.Phillipot : *Manuel utilisateur Screen Maker*, Logiciel et Médias, Paris, janvier 1987.

G.Cochonneau : *Manuel d'utilisation de Masque.*

N.Domingues : *MicroEditor, manual do usuario.*

Domingues, N. & Séchet, P. : *Ferramentas para implementação de aplicações "tela-cheia".*