Un environnement conversationnel à deux dimensions

B. Meyer
EDF/DER-IMA, Avenue du Général de Gaulle
92141 Clamart Cédex

1 - INTRODUCTION

Dans toutes les branches d'application de l'informatique, l'utilisation de moyens conversationnels se généralise. L'évolution constatée aujourd'hui ne se limite pas au fait que le mode de soumission des travaux en "traitement par lots" cède de plus en plus souvent la place à un dialogue interactif : elle affecte aussi la forme même de ce dialogue, qui s'effectuait en mode dit "ligne à ligne" sur les machines à écrire et les premières consoles à rayons cathodiques, alors que les terminaux modernes utilisent comme unité de communication avec l'ordinateur le contenu d'un écran entier ; c'est ce qu'on appelle le dialogue en mode "plein écran" ou "pleine page".

L'une des utilisations les mieux connues de cette technique est la préparation de textes sur ordinateur, à l'aide d'un "éditeur pleine page" tel qu'il en existe aujourd'hui sur un certain nombre de matériels, en particulier sur les ordinateurs de grande puissance et les machines de bureautique . Les utilisateurs sont unanimes à reconnaître la grande efficacité de ces outils, qui rend extrêmement désagréable le retour ultérieur à un éditeur classique. Le dialogue pleine page trouve aujourd'hui des applications de plus en plus fréquentes dans d'autres domaines, comme la mise au point de logiciel, ou encore l'utilisation de programmes par des non-spécialistes sous le contrôle de "menus" successifs ; parmi les applications qui ont été le plus généralement gagnées par cette dernière idée, on peut citer la gestion ("systèmes transactionnels") et l'Enseignement Assisté par Ordinateur. De plus en plus nombreux sont les programmeurs qui, dans tous les domaines, cherchent à utiliser les mêmes techniques pour l'exécution de leurs programmes.

La construction de dialogues "pleine page" ajoute aux problèmes généraux de la programmation conversationnelle, qui sont loin d'être parfaitement maîtrisés (en particulier pour ce qui est de l'ergonomie des dialogues), la difficulté supplémentaire née de l'utilisation de textes en deux dimensions, tels qu'ils apparaissent sur l'écran. L'objet de cet article est d'étudier quelques-uns de ces problèmes et de décrire quelques-unes des solutions mises en place à la Direction des Etudes et Recherches d'EDF. La discussion se rapporte aux trois composantes fondamentales du logiciel : les méthodes, les outils, les langages ; un autre aspect important, l'ergonomie des dialogues, n'est abordé que brièvement.

On pourra juger à juste titre que ces réflexions sont en retrait sur l'évolution des techniques matérielles et logicielles les plus avancées. Nous nous limitons en particulier à la manipulation d'objets textuels, alors qu'une expérience considérable a été acquise depuis de nombreuses années dans deux domaines voisins, les systèmes graphiques et la Conception Assistée par Ordinateur. Il est clair par ailleurs que certains laboratoires de recherche ont mis au point des environnements à deux dimensions plus perfectionnés que celui qui est décrit ici : on peut ainsi citer les nombreux outils développés autour de LISP /10/, ainsi que SMALLTALK, mis au point au laboratoire de recherche Xerox de Palo Alto /2/, qui utilise des terminaux spéciaux et un système d'exploitation sur mesure. Mais les outils décrits ci-après, et les idées qui ont présidé à leur conception, paraissent encore bien peu courants dans les environnements "ordinaires", industriels ou universitaires. On notera ainsi que les Communications de l'ACM ont publié récemment deux articles /3, 7/ sur les problèmes de la programmation interactive ; forts différents par ailleurs, ces deux articles cherchent l'un et l'autre à définir la meilleure façon de présenter aux utilisateurs les questions successives, de trouver des mots-clés mnémoniques, de traiter les erreurs, etc. ; or tous deux

se placent d'emblée dans le cas où le dialogue est strictement séquentiel, sans imaginer qu'il puisse en exister d'autre. Une bonne partie de la discussion contenue dans ces articles devient sans objet lorsqu'on passe à des environnements à deux dimensions.

L'environnement présenté dans cet article a précisément pour particularité de s'insérer dans un centre de calcul de type classique, à base de matériels IBM (3081, 3033, 370-168, 4341, etc.) sous le système d'exploitation MVS-SP. Le système conversationnel est TSO. Les terminaux "pleine page" appartiennent à la série 3270 ; ce sont essentiellement des 3278 et des 3279 modèles 2B et 3B (ces derniers possédant sept couleurs, des possibilités "semi-graphiques" et diverses autres options). L'essentiel de la programmation des applications se fait en Fortran.

2 - CARACTERISTIQUES DES DIALOGUES A DEUX DIMENSIONS

L'intérêt des dialogues à deux dimensions vient de la combinaison de trois propriétés :

- La deuxième dimension proprement dite : l'utilisateur du programme jouit à chaque instant d'une vue s'étendant sur toute une page de texte, et non pas seulement sur une ligne ;

- L'emploi de la page comme unité d'échange avec l'ordinateur, qui permet à l'utilisateur de composer un tableau d'ensemble, de corriger les erreurs éventuelles, d'hésiter et de revenir sur ses décisions, avant de transmettre une série de commandes ou d'informations ;

- La possibilité pour le programme de remplir à l'avance certaines zones en y insérant des valeurs par défaut, que l'utilisateur modifiera seulement si c'est nécessaire : on lui évitera ainsi de répondre à des questions qui n'ont pas de sens dans son cas particulier, ou appellent la même réponse d'une utilisation à la suivante (un des reproches les plus fréquemment adressés aux programmes conversationnels qui n'ont pas cette propriété est qu'il faut "raconter sa vie" à chaque utilisation).

Un bon programme en mode page conservera pour chacun de ses utilisateurs un profil qui permettra de proposer pour chaque question, comme réponse par défaut, non pas un choix toujours identique, mais la dernière réponse fournie lors des utilisations précédentes.

A titre d'exemple, on trouvera ci-après un exemple de dialogue en mode page ; il s'agit des premières étapes de l'utilisation de la procédure Fortran (compilation, reliure, exécution) de la bibliothèque "AL" mise en place dans notre centre de calcul (cf. 4 ci-après). On peut imaginer le nombre de questions successives qui seraient nécessaires pour obtenir le même résultat dans un dialogue ligne à ligne ; la plupart des réponses seraient identiques d'une utilisation à la suivante. Sans le mode page, le concepteur d'un tel dialogue est sans cesse écartelé entre les utilisateurs "spéciaux", qui, voulant pouvoir accéder à un grand nombre de possibilités, souhaitent qu'on leur pose beaucoup de questions, et les utilisateurs "vulgaires", les plus nombreux, qui emploient les options les plus courantes et sont soucieux de la brièveté du dialogue.

On notera la présence d'une option intitulée "la même chose que la dernière fois" qui (de façon semblable à ce qu'on peut demander chez son coiffeur attitré) permet d'être complètement silencieux et de ne se voir poser aucune question ; cette option est particulièrement utile lors d'un travail répétitif tel que la mise au point d'un module.

```
- AL -----------------PROCEDURE FORTRAN----------------------------- AL -
            En cas de difficultes, appuyer sur HELP (FP1)

     BONJOUR, BERTRAND

          COCHER L'OPTION CHOISIE :

     1- meme chose que la derniere fois..... ===>

     2- compilation, edition de liens, execution.. ===> X

     3- edition de liens, execution..... ===>

     4- execution..... ===>
```

```
- AL -----------------COMPILATION FORTRAN H EXTENDED----------------- AL -
            En cas de difficultes, appuyer sur HELP (FP1)

NOM DU FICHIER A COMPILER :
  ===> TENTATIV.FORT(NUMERO1)

IMPRESSION DE LA LISTE :                          (TER,IMP,LOC,DMY,SYS
  ===> IMP                                         ou un NOM DE FICHIER)

CLASSE :                                          (si SYS: C,R,S ou U)
  ===>

DESTINATION :                                     (si IMP, LOC ou
  ===> C                                           SYS classe C )

NOM DU FICHIER QUI CONTIENDRA LE MODULE OBJET :
  ===> 'TTHHHDA.FORTRAN.OBJ'

OPTIONS DU COMPILATEUR :                          ( 0 ou 2 )
  ===> 0                                          (LIST ou NOLIST)
  ===> NOLIST
```

```
TERMINAL DISPONIBLE
- AL ------------------ EDITION DE LIENS ------------------------- AL -
            En cas de difficultes,appuyer sur HELP (FP1)

   Vous pouvez appeler des bibliotheques en entrant :
        -Un mot-cle (FORTLIB,GENERALE,IMSL,LINPACK,BENSON,ATELBIB...)
        -Un nom de fichier contenant une bibliotheque
     ===> FORTLIB

     ===>

     ===>

     ===>

     ===>

     ===>

     ===> GENERALE

   NOM DU FICHIER CONTENANT LE MODULE OBJET :
     ===> 'TTHHHDA.FORTRAN.OBJ'
   NOM DU FICHIER QUI CONTIENDRA LE MODULE EXECUTABLE :
        'LHH9092.FORTRAN.LOAD'     (attribue par la procedure)
```

(etc.)

On peut affirmer sans exagération que, pour le programmeur qui prépare des dialogues de ce type, le saut est aussi grand d'un programme conversationnel simple (ligne à ligne) à un programme conversationnel en mode page que d'un programme non conversationnel ("batch") à un programme conversationnel simple. Nous parlerons à ce propos, avec quelque emphase, d'"informatique à deux dimensions" ; la seconde dimension, verticale, introduite par le mode page exige une sérieuse réflexion sur les méthodes et les techniques de l'interactivité, ainsi que des outils adéquats.

3 - PROCEDURES DE COMMANDE : LE GESTIONNAIPF DE DIALOGUES

Le premier outil qui s'offre aux utilisateurs est fourni par le constructeur : IBM a diffusé au printemps dernier une nouvelle version de SPF (System Productivity Facility /5/), un sous-système conversationnel de TSO (le système conversationnel de base). SPF offre une grande facilité d'utilisation grâce à trois caractéristiques principales :

- l'utilisation des deux dimensions ;
- la possibilité de conserver d'une session à la suivante certaines caractéristiques de l'utilisateur, et de lui éviter ainsi le codage répété de renseignements fixes ;
- un mode particulier de traitement des erreurs.

Un écran typique de SPF est donné ci-dessous ("menu" de l'édition d'un fichier). Les trois caractéristiques précédentes y apparaissent : deux dimensions, options préremplies par le système en fonction des demandes antérieures, message d'erreur.

```
------------------------- EDIT - ENTRY PANEL ---------- DATASET NOT CATALOGED
ENTER/VERIFY PARAMETERS BELOW:

SPF LIBRARY:
   PROJECT ===> FICHIER  } <- préremplis
   LIBRARY ===> KINEXIST } ===>          ===>          ===>
   TYPE    ===> PAS
   MEMBER  ===>          (BLANK FOR MEMBER SELECTION LIST)

OTHER PARTITIONED OR SEQUENTIAL DATASET:
   DATASET NAME  ===>
   VOLUME SERIAL ===>          (IF NOT CATALOGED)

DATASET PASSWORD ===>          (IF PASSWORD PROTECTED)

PROFILE NAME     ===>          (BLANK DEFAULTS TO DATASET TYPE)
```

(message)

L'apport majeur de la nouvelle version de SPF est l'ensemble de fonctions présenté sous le nom de "Gestionnaire de Dialogues" (Dialog Manager /6/). Le gestionnaire de dialogues rend accessible à n'importe quel utilisateur, pour l'écriture de ses propres procédures, les outils employés de façon interne par SPF ; il permet donc de généraliser à une procédure de commande quelconque les trois propriétés décrites ci-dessus.

Concrètement, le gestionnaire de dialogues peut être appelé par les procédures écrites dans le langage de commande de TSO, qui a été étendu de façon à permettre la définition d'écrans ("menus" ou "panels" dans la terminologie de SPF), la création et la mise à jour de "tables" (qui sont des fichiers d'un type particulier, accessibles simultanément à plusieurs utilisateurs et pouvant en particulier servir à conserver des informations rémanentes) et le traitement des erreurs. Il est cependant d'un maniement assez ardu, et non destiné à des utilisateurs non spécialistes. Par ailleurs, le Gestionnaire de Dialogues ne peut pas (ou peut difficilement) être appelé par un programme d'application, écrit en Fortran par exemple. Son utilisation majeure dans notre équipe a été jusqu'ici l'écriture d'une bibliothèque de procédures de commande d'intérêt général, la bibliothèque AL.

4 -LA BIBLIOTHEQUE AL

La bibliothèque AL (Atelier Logiciel) de procédures TSO contient actuellement une quarantaine de procédures qui ouvrent un large éventail de possibilités : accès aux compilateurs des différents langages disponibles (de Fortran à Cobol en passant par le langage d'assemblage, Algol W, Pascal, Simula 67, Reduce), manipulations de fichiers, utilisation de programmes spécialisés, accès à la documentation, etc.

Les procédures de la bibliothèque AL s'exécutaient jusqu'ici en mode "ligne à ligne", avec les inconvénients ci-dessus mentionnés. Le souci de conserver des procédures simples, et donc de limiter le nombre d'options, avait eu pour conséquence la prolifération de versions "personnalisées" des procédures les plus importantes, recopiées puis modifiées par certains programmeurs à partir de la version "officielle". Une telle situation n'était évidemment pas acceptable, ne serait-ce que du fait des problèmes liés à l'évolution des procédures "piratées", qu'il n'était pas possible de faire bénéficier des améliorations successives apportées à la version de référence.

Avec le passage en deux dimensions, ces problèmes disparaissent : à la première utilisation d'une procédure, on doit bien donner tous les détails, mais le système s'en souviendra d'une session à la suivante et les réaffichera : seules les options qui ont changé devront être refrappées. Comme, en général, les paramètres d'exécution varient peu d'un appel au suivant, la situation est beaucoup plus confortable pour les utilisateurs, d'autant plus que l'utilisation des écrans complets (24 lignes x 80 colonnes ou plus) permet d'inclure toutes les options intéressantes. Toutes les procédures proposent en outre initialement l'option appelée "la même chose que la dernière fois" vue plus haut.

Les procédures actuellement disponibles sous cette forme sont Fortran (IV), Fortran VS, Simula, Pascal, Algol W, Cobol, Apothèce (outil de gestion de bibliothèques de programmes). La bibliothèque AL sera progressivement adaptée tout entière.

5 - DES PROGRAMMES D'APPLICATION EN DEUX DIMENSIONS : GESCRAN

Lorsqu'on a découvert, par exemple à travers l'utilisation de SPF et des procédures AL, les délices des deux dimensions, il est tentant d'en faire bénéficier ses propres programmes, et de repenser en conséquence les dialogues qu'ils effectuent avec leurs utilisateurs. Un progiciel a été développé à cet effet : Gescran /1/, un ensemble de sous-programmes construits selon les principes décrits dans /9/. Gescran permet à un programme quelconque, par une série d'appels à des sous-programmes Fortran, de définir un ensemble d'objets abstraits appelés "écrans", de créer dans ces écrans un certain nombre de "fenêtres" rectangulaires, d'initialiser le contenu de ces fenêtres, de spécifier et de modifier leurs caractéristiques (brillance, protection, couleur, etc.), d'afficher les "écrans" sur un terminal, et de prendre en compte les données de différents types frappées par l'utilisateur de ce terminal. Insistons sur le fait que les écrans et les fenêtres sont des objets purement abstraits, connus du programmeur seulement par leur nom qui est, en Fortran, une variable entière (utilisée de façon interne pour contenir une adresse et diverses autres informations), à laquelle peuvent seulement être appliquée les manipulations effectuées par les sous-programmes de Gescran.

Gescran a été écrit pour fonctionner sur les 3270 et compatibles, mais il est conçu pour pouvoir être adapté à tout type de terminal offrant des possibilités équivalentes ; la construction et la manipulation des structures de données représentant les écrans et leurs fenêtres sont entièrement disjointes des opérations d'entrée et de sortie physiques.

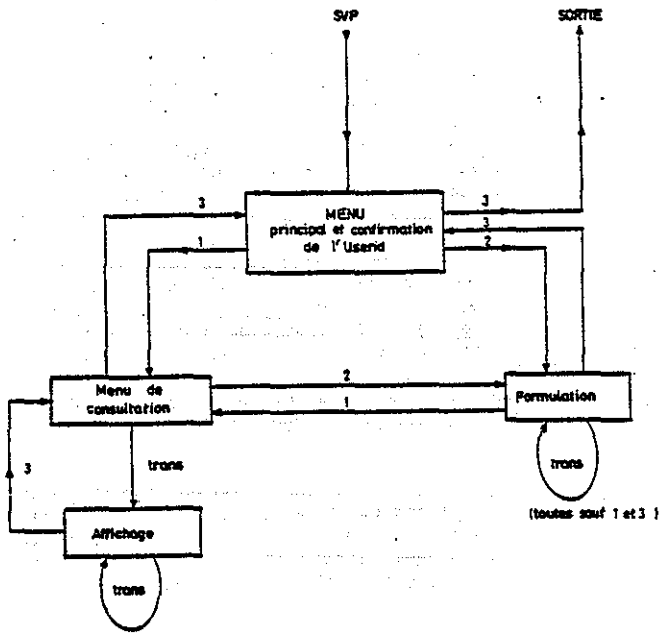
Parmi les compléments qui sont en cours de mise en oeuvre, nous signalerons en particulier l'extension de la notion d'écran, permettant de dissocier totalement la taille d'un écran abstrait de celle des terminaux effectivement utilisés. Nous espérons également pouvoir coupler Gescran avec un logiciel graphique, en offrant au programmeur la faculté de définir une certaine fenêtre comme graphique, si, bien entendu, le terminal possède les propriétés correspondantes.

6 - CAO D'ECRANS : CONSCRAN

Un outil important pour l'utilisation efficace de Cescran sera prochainement diffusé ; il s'agit de Conscran, utilitaire de construction d'écrans. Conscran répond à un problème qu'ont rencontré tous les utilisateurs de Gescran : avant de pouvoir écrire la suite d'appels de sous-programmes qui demandera à Gescran la construction d'un écran, il faut concevoir celui-ci, c'est-à-dire définir la position de ses différentes fenêtres, leur rôle, les textes initiaux qu'elles devront contenir, leurs caractéristiques de couleur, de protection, etc. La seule façon pratique d'effectuer cette opération est de dessiner cet écran sur une feuille de papier quadrillé où l'on essaiera les différentes combinaisons (l'image la plus proche étant celle du typographe qui compose une page de journal en combinant différents "pavés"). L'emploi d'un tel support peut paraître quelque peu primitif en regard du but poursuivi. Conscran permet d'effectuer les mêmes opérations de façon interactive, et en deux dimensions : on fabrique son écran à la console, avec toute la souplesse qui en résulte. Le programme Conscran est lui-même entièrement interactif et fonctionne en mode page ; il permet des retours en arrière, l'archivage de la structure de données décrivant une série d'écrans en vue de sa modification ultérieure, etc. Il s'agit donc d'une véritable "CAO d'écrans" (CAE ?). Bien entendu, le programme Conscran est une application à deux dimensions, écrite à l'aide de Gescran.

7 - SUR LA STRUCTURE DES PROGRAMMES DE DIALOGUE

Les meilleurs outils du monde (dont nous ne prétendons par qu'ils soient ceux qui viennent d'être décrits) ne sont pas suffisants pour programmer efficacement en deux dimensions. Un point délicat est la structure d'ensemble des programmes de dialogue. Leur comportement peut en général être assez fidèlement modélisé par un diagramme de transition ; une exécution du programme conversationnel est représentée par un certain cheminement dans le graphe correspondant. On trouvera ci-après un exemple très simple de graphe de ce type ; il s'agit d'une application réalisée à l'aide de Gescran, le système SVP /4/, qui permet de dialoguer en différé avec le service d'assistance aux utilisateurs. Seul le volet "questionneur" est représenté.



A sa simplicité près, cet exemple est très représentatif de la structure des programmes conversationnels en mode page. Chaque étape du dialogue, associée à l'un des états du diagramme, correspond à l'affichage d'un écran par le programme, suivi du remplissage par l'utilisateur d'un certain nombre de zones et d'une série de contrôles effectués par le programme. La voie choisie pour la suite du dialogue dépend des réponses de

l'utilisateur ; dans cet exemple, on se voit proposer à la fin de chaque étape une série d'options possibles, désignées par des numéros (représentés dans le graphe par les étiquettes des différents arcs), qui déterminent le passage à l'état suivant. Nous supposerons par la suite qu'il en est toujours ainsi. En pratique, sur notre système, l'utilisateur choisit un numéro en appuyant sur l'une des "touches de fonction" du clavier (ou sur la touche "trans", correspondant au numéro zéro).

Dans ce genre d'applications, la structure d'une étape de base est la suivante :

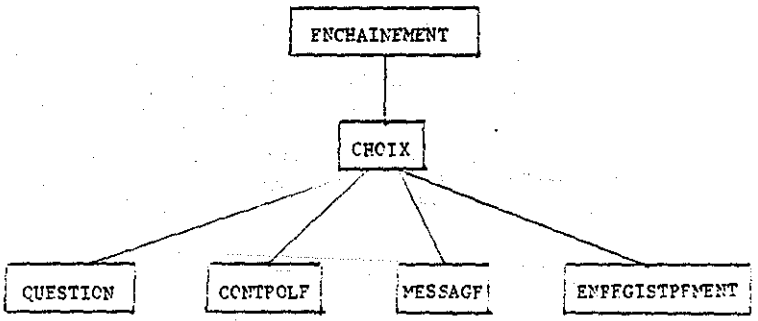
```
étape x :
        afficher l'écran x ;
        répéter
                lire les réponses de l'utilisateur et examiner la
                touche t utilisée pour les renvoyer ;
                si erreur dans les données alors
                        afficher un message
        jusqu'à pas d'erreur dans les données ;
        prendre en compte les réponses ;

        si      t = t1 alors passer à l'étape x1
        sinon si t = t2 alors passer à l'étape x2
        sinon si  ... (etc.)
```

Une programmation directement déduite de cette formulation donne des programmes à branchements inextricables, du type "plat de spaghettis" bien connu. Ceci est dû au fait que le graphe des écrans est de structure arbitraire ; en particulier, il est en général nécessaire de prévoir des retours en arrière, des détours temporaires (touches "aide") et une option qui permet d'abandonner le dialogue à un instant quelconque de son déroulement (ces deux dernières possibilités ne sont pas présentes dans le graphe précédent). Ces contraintes mettent en défaut la "programmation structurée" dans son acception vulgaire.

La solution qui s'impose ici est celle de la "commande par table", qui se rapproche des tables de décision utilisées en gestion et des automates bien connus en commande de processus et en compilation. Elle considère le programme conversationnel comme un système à états, dont l'évolution consiste en une série de transitions ; chaque transition est déterminée par l'état d'origine et par le numéro de la touche de fonction utilisée pour renvoyer les réponses. Les graphes utilisés en pratique pouvant être arbitrairement complexes, et pouvant évoluer de façon considérable au cours du cycle de vie du programme (en particulier sous l'influence des utilisateurs), il est tout à fait indiqué de les représenter, non dans la structure du programme, mais dans celle des données, sous la forme d'une table.

Plus précisément, on utilisera quatre unités de programme distinctes sur trois niveaux hiérarchiques :



Le programme ENCHAINEMENT définit seulement le cheminement dans le graphe ; il ne connaît rien des écrans propres à l'application (et doit pouvoir être réutilisé à l'identique pour des applications différentes) :

```
ENCHAINEMENT :
        état := état_initial ;
        répéter
                t := CHOIX (état) ;
                état := transition (état, t)
        jusqu'à état = état_final
```

"transition" est la fonction qui décrit le graphe des états : transition (e, t) est l'état obtenu lorsqu'on quitte l'état e par la voie t (t est dans notre modèle le numéro d'une touche de fonction). En suivant le principe énoncé précédemment, on représentera transition par un tableau à deux dimensions (matrice des transitions) : le programme est alors insensible aux changements de politique de cheminement. En opérant ainsi, on définit un graphe "programmable" plutôt que de le "câbler" une fois pour toutes.

Le sous-programme CHOIX a pour tâche d'opérer un aiguillage entre les différents écrans de l'application, de garantir que la réponse de l'utilisateur fait partie du répertoire officiel et de renvoyer le numéro de la touche de sortie :

```
CHOIX (i) :
        répéter
                (r, t) := QUESTION (i) ;
                correct := CONTROLE (r, t, i) ;
                si non correct alors
                        MESSAGE (r, t, i)
        jusqu'à correct
        ENREGISTREMENT (r, i) ;
        renvoyer le numéro t de la touche de fonction utilisée
```

On notera que CHOIX peut lui aussi être programmé de façon indépendante de l'application si l'on adopte une description suffisamment abstraite des objets de type "réponse".

QUESTION, CONTROLE, MESSAGE et ENREGISTREMENT sont, eux, spécifiques de l'application traitée. Le premier décrit ce qu'on peut appeler sa syntaxe, les autres, sa sémantique. L'appel QUESTION (i) permet l'association entre un numéro d'état et l'écran correspondant ; il affiche cet écran et lit les réponses de l'utilisateur du programme aux questions posées :

```
QUESTION (i) :
        cas n° i parmi
                1 : afficher l'écran numéro 1,
                2 : afficher l'écran numéro 2,
                ..............................
                ..............................
                n : afficher l'écran numéro n ;
        lire les réponses r et la touche de fonction t ;
        renvoyer (r, t)
```

CONTROLE (r, t, i) vérifie la validité de la réponse (r, t) pour l'état i ; MESSAGE (r, t, i) affiche si nécessaire un message d'erreur ; enfin, ENREGISTREMENT (r, i) effectue les actions résultant de la prise en compte de la réponse r dans l'état i. Très souvent, il s'agira de la mise à jour d'une base de données.

Il serait évidemment souhaitable de pouvoir disposer d'outils capables de construire les programmes de dialogue en partant d'une description des six ingrédients ci-dessus exprimée dans un langage simple. Nous ne connaissons pas d'outil vraiment général dans ce domaine ; les développements qui s'en rapprochent le plus sont ceux des "langages d'auteurs" en enseignement assisté par ordinateur (IMG, Plato, etc.), et certains outils d'aide à la construction de systèmes transactionnels de gestion.

8 - UN LANGAGE BIEN ADAPTE : SIMULA

Pour la mise en oeuvre des principes méthodologiques précédents, l'utilisation du langage Simula 67 nous a paru particulièrement efficace. Les notions fondamentales sont celles sur lesquelles l'accent a été mis dans /8/ : types abstraits, conception descendante des programmes et des structures de données, généricité.

Pour programmer en pratique le schéma précédent, il est en effet très utile de pouvoir disposer d'une structure correspondant à la notion abstraite d'état. A tout état e sont associées un certain nombre de caractéristiques :

- attributs de l'état e : numéro associé à e, écran devant être affiché lorsqu'on atteint cet état ;
- opérations pouvant être demandées lorsque le système se trouve dans l'état e : QUESTION, CONTROLE, MESSAGE, ENREGISTREMENT ;
- actions devant être effectuées lorsqu'on atteint l'état e : CHOIX.

Ces différentes caractéristiques correspondent exactement à ce qu'on peut inclure dans la structure de base offerte par le langage Simula, la classe, représentation d'un type d'objets abstraits : des variables représentant les attributs de chaque état, des sous-programmes (procédures) correspondant aux opérations admissibles, et des instructions correspondant aux actions de création. On est donc tout naturellement amené à définir une classe ETAT à laquelle s'appliquent les sous-programmes ENCHAINEMENT, CHOIX, QUESTION etc.

Une propriété fondamentale de Simula est ici la notion de préfixation de classe et celle de procédure virtuelle qui la complète. Elles offrent la possibilité d'une construction véritablement descendante et modulaire. On définira la classe générique ETAT selon le schéma ci-après :

```
class ETAT ;
      begin

      comment attributs :                ;
            integer numéro_associé ;
            integer écran_associé ; comment Rappelons que pour
                                    Gescran un écran est
                                    défini par une
                                    variable entière ;

      comment opérations :               ;
            virtual :
                  ref (réponse) procedure QUESTION ;
                  procedure CONTROLE ;
                  procedure MESSAGE ;
                  procedure ENREGISTREMENT ;

      comment actions :
;
            CHOIX (numéro_associé) ;

      end  (class ETAT)
```

La classe ETAT définit les propriétés générales d'un écran. Les sous- programmes ENCHAINEMENT et CHOIX peuvent être complètement écrits à l'aide de cette classe ; par exemple, si l'on utilise, comme nous le suggérons, la commande par table, le tableau transition sera du type générique ETAT (ref (ETAT) en Simula).

Pour la programmation d'une application particulière, on précisera ("instanciera") la classe ETAT en décrivant des classes préfixées par celle-ci :

```
ETAT class MENU_INITIAL ; begin  ...  end ;
ETAT class OPTIONS_DE_COMPILATION ; begin  ...  end ;
etc.
```

Dans chacune de ces sous-classes, on inclura le corps des procédures QUESTION, CONTROLE, MESSAGE, ENREGISTREMENT propres à l'état considéré ; au niveau de la classe générique ETAT, ces procédures "virtuelles" sont connues seulement par leur spécification partielle.

Le grand avantage de cette méthode est qu'elle permet une construction véritablement modulaire : la partie indépendante de l'application est programmée séparément ; pour une application donnée, chaque état est ensuite décrit de façon entièrement autonome, sans référence à l'enchaînement du dialogue, mais avec toutes les caractéristiques de l'état sans exception : écran à afficher, réponses à lire, contrôles de validité, messages d'erreur, enregistrement des réponses validées.

Ce mode de programmation modulaire, particulièrement élégant, n'a pas d'équivalent dans les autres langages de programmation.

9 - CONCLUSION

Nous avons décrit un ensemble d'outils et de réflexions méthodologiques encore récent, et en constante évolution. Nous souhaitons que les ambitieux développements actuels en matière d'"environnements de programmation" prennent en compte l'aspect bidimensionnel développé dans cet article, et ne négligent pas les problèmes ergonomiques que soulève, dans le domaine du développement de logiciel comme ailleurs, la mise en oeuvre d'une communication homme-machine réussie.

BIBLIOGRAPHIE

/1/ E. Audin, G. Brisson, P. Meyer : Gescran ; note EDF, Atelier Logiciel n° 22, décembre 1980 (version 4, décembre 1981).

/2/ BYTE Magazine : Numéro spécial sur SMALLTALK, août 1981.

/3/ B. Dwyer : A User-Friendly Algorithm ; Communications of the ACM, 24, 9, pp. 556-561, septembre 1981.

/4/ E. de Drouas : Manuel d'Utilisation de SVP ; note EDF, Atelier Logiciel n° 32, octobre 1981.

/5/ IBM : System Productivity Facility for MVS - Program Reference ; order no. SC34-2038-0, décembre 1980.

/6/ IBM : System Productivity Facility for MVS - Dialog Management Services ; order no. SC34-2036-1, mars 1981.

/7/ H. Ledgard, J.A. Whiteside, A. Singer, W. Seymour : The Natural Language of Interactive Systems ; Communications of the ACM, 23, 10, pp. 556-563, octobre 1980.

/8/ B. Meyer : Quelques Concepts des Langages de Programmation modernes, et leur Application à SIMULA 67 ; Bulletin AFCET-GPCPLAN n° 9, 1979 (également note EDF Atelier Logiciel n° 15).

/9/ B. Meyer : Principles of Package Design (Principes de Conception de Progiciels) ; Note EDF Atelier Logiciel n° 29, avril 1981 (à paraître dans CACM).

/10/ W. Teitelman : A Display Oriented Programmer's Assistant, in Proc. 5th Int. Jt. Conf. on Artificial Intelligence, Dpt. Comp. Sc., Carnegie-Mellon Univ., Pittsburgh, 1977, pp. 905-915 (cf. aussi E. Sandewall : Programming in the Interactive Environment : The LISP Experience, ACM Comp. Surv., 10, 1, mars 1978, pp. 35-72).