N° d'ordre : 1073

THÈSE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir

LE TITRE DE DOCTEUR DE 3ème CYCLE

INFORMATIQUE

par

Bertrand PETITPREZ

MODELE ARBORESCENT DE DESCRIPTION DE LOGICIELS INTERACTIFS FACILITANT LES REPRISES.

Thèse soutenue le 5 Septembre 1983 devant la Commission d'Examen

Membres du Jury :

Président	G. COMYN
Rapporteur	C. CARREZ
Examinateurs	B. DRIEUX
	B. MEYER
	R. ROUSSEAU

PROFESSEURS 1ère CLASSE

M.	BACCHUS Pierre	Mathématiques
M.	BEAUFILS Jean-Pierre (dét.)	Chimie
M.	BIAYS Pierre	G.A.S.
M.	BILLARD Jean	Physique
M.	BONNOT Ernest	Biologie
M.	BOUGHON Pierre	Mathématiques
M.	BOURIQUET Robert	Biologie
M.	CELET Paul	Sciences de la Terre
M.	COEURE Gérard	Mathématiques
M.	CONSTANT Eugène	I.E.E.A.
M.	CORDONNIER Vincent	I.E.E.A.
M.	DEBOURSE Jean-Pierre	S.E.S.
M.	DELATTRE Charles	Sciences de la Terre
M.	DURCHON Maurice	Biologie
M.	ESCAIG Bertrand	Physique
M.	FAURE Robert	Mathématiques
M.	FOURET René	Physique
M.	GABILLARD Robert	I.E.E.A.
M.	GRANELLE Jean-Jacques	S.E.S.
M.	GRUSON Laurent	Mathématiques
M.	GUILLAUME Jean	Biologie
M.	HECTOR Joseph	Mathématiques
M.	HEUBEL Joseph	Chimie
M.	LABLACHE COMBIER Alain	Chimie
M.	LACOSTE Louis	Biologie
M.	LANSRAUX Guy	Physique
M.	LAVEINE Jean-Pierre	Sciences de la Terre
M.	LEBRUN André	C.U.E.E.P.
M.	LEHMANN Daniel	Mathématiques
Mme	LENOBLE Jacqueline	Physique
M.	LHOMME Jean	Chimie
M.	LOMBARD Jacques	S.E.S.
M.	LOUCHEUX Claude	Chimie
M.	LUCQUIN Michel	Chimie

M.	MAILLET Pierre	S.E.S.
M.	MONTREUIL Jean	Biologie
M.	PAQUET Jacques	Sciences de la Terre
M.	PARREAU Michel	Mathématiques
M.	PROUVOST Jean	Sciences de la Terre
M.	SALMER Georges	I.E.E.A.
Mme	SCHWARTZ Marie-Hélène	Mathématiques
M.	SEGUIER Guy	I.E.E.A.
M.	STANKIEWICZ François	Sciences Economiques
M.	TILLIEU Jacques	Physique
M.	TRIDOT Gabriel	Chimie
M.	VIDAL Pierre	I.E.E.A.
M.	VIVIER Emile	Biologie
M.	WERTHEIMER Raymond	Physique
M.	ZEYTOUNIAN Radyadour	Mathématiques

PROFESSEURS 2ème CLASSE

M.	AL FAKIR Sabah	Mathématiques
M.	ANTOINE Philippe	Mathématiques
M.	BART André	Biologie
Mme	BATTIAU Yvonne	Géographie
M.	BEGUIN Paul	Mathématiques
M.	BELLET Jean	Physique
M.	BKOUCHE Rudolphe	Mathématiques
M.	BOBE Bernard	S.E.S.
M.	BODART Marcel	Biologie
M.	BOILLY Bénoni	Biologie
M.	BONNELLE Jean-Pierre	Chimie
M.	BOSQ Denis	Mathématiques
M.	BREZINSKI Claude	I.E.E.A.
M.	BRUYELLE Pierre (Chargé d'enseignement)	Géographie
M.	CAPURON Alfred	Biologie
M.	CARREZ Christian	I.E.E.A.
M.	CHAMLEY Hervé	E.U.D.I.L.

M. CHAPOTON Alain	C.U.E.E.P.
M. COQUERY Jean-Marie	Biologie
Mme CORSIN Paule	Sciences de la Terre
M. CORTOIS Jean	Physique
M. COUTURIER Daniel	Chimie
Mle DACHARRY Monique	Géographie
M. DEBRABANT Pierre	E.U.D.I.L.
M. DEGAUQUE Pierre	I.E.E.A.
M. DELORME Pierre	Biologie
M. DEMUNTER Paul	C.U.E.E.P.
M. DE PARIS Jean-Claude	Mathématiques
M. DEVRAINNE Pierre	Chimie
M. DHAINAUT André	Biologie
M. DORMARD Serge	S.E.S.
M. DOUKHAN Jean-Claude	E.U.D.I.L.
M. DUBOIS Henri	Physique
M. DUBRULLE Alain	Physique
M. DUEE Gérard	Sciences de la Terre
M. DYMENT Arthur	Mathématiques
M. FLAMME Jean-Marie	E.U.D.I.L.
M. FONTAINE Hubert	Physique
M. GERVAIS Michel	S.E.S.
M. GOBLOT Rémi	Mathématiques
M. GOSSELIN Gabriel	S.E.S.
M. GOUDMAND Pierre	Chimie
M. GREVET Patrice	S.E.S.
M. GUILBAULT Pierre	Biologie
M. HANGAN Théodore	Mathématiques
M. HERMAN Maurice	Physique
M. JACOB Gérard	I.E.E.A.
M. JACOB Pierre	Mathématiques
M. JOURNEL Gérard	E.U.D.I.L.
M. KREMBEL Jean	Biologie
M. LAURENT François	I.E.E.A.
Mle LEGRAND Denise	Mathématiques
Mle LEGRAND Solange	Mathématiques (Calais)
Mme LEHMANN Josiane	Mathématiques

M. LEMAIRE Jean	Physique
M. LENTACKER Firmin	G.A.S.
M. LEVASSEUR Michel	I.P.A.
M. LHENAFF René	G.A.S.
M. LOCQUENEUX Robert	Physique
M. LOSFELD Joseph	I.E.E.A.
M. LOUAGE Francis	E.U.D.I.L.
M. MACKE Bruno	Physique
M. MAIZIERES Christian	I.E.E.A.
Mle MARQUET Simone	Mathématiques
M. MESSELYN Jean	Physique
M. MIGEON Michel	E.U.D.I.L.
M. MIGNOT Fulbert	Mathématiques
M. MONTEL Marc	Physique
Mme NGUYEN VAN CHI Régine	G.A.S.
M. PARSY Fernand	Mathématiques
Mle PAUPARDIN Colette	Biologie
M. PERROT Pierre	Chimie
M. PERTUZON Emile	Biologie
M. PONSOLLE Louis	Chimie
M. PORCHET Maurice	Biologie
M. POVY Lucien	E.U.D.I.L.
M. RACZY Ladislas	I.E.E.A.
M. RICHARD Alain	Biologie
M. RIETSCH François	E.U.D.I.L.
M. ROGALSKI Marc	M.P.A.
M. ROUSSEAU Jean-Paul	Biologie
M. ROY Jean-Claude	Biologie
M. SALAMA Pierre	S.E.S.
Mme SCHWARZBACH Yvette (CCP)	M.P.A.
M. SCHAMPS Joël	Physique
M. SIMON Michel	S.E.S.
M. SLIWA Henri	Chimie
M. SOMME Jean	G.A.S.
Mle SPIK Geneviève	Biologie
M. STERBOUL François	E.U.D.I.L.
M. TAILLIEZ Roger	Institut Agricole

M.	TOULOTTE Jean-Marc	I.E.E.A.
M.	VANDORPE Bernard	E.U.D.I.L.
M.	WALLART Francis	Chimie
M.	WATERLOT Michel	Sciences de la Terre
Mme	ZINN JUSTIN Nicole	M.P.A.

CHARGES DE COURS

M.	TOP Gérard	S.E.S.
M.	ADAM Michel	S.E.S.

CHARGES DE CONFERENCES

M.	DUVEAU Jacques	S.E.S.
M.	HOFLACK Jacques	I.P..A
M.	LATOUCHE Serge	S.E.S.
M.	MALAUSSENA DE PERNO Jean-Louis	S.E.S.
M.	OPIGEZ Philippe	S.E.S.

Je tiens tout d'abord à remercier Christian Carrez de l'estime qu'il m'a accordée en me confiant un problème qui lui tenait à cœur depuis des années. Ce travail doit beaucoup aux recherches qu'il a menées en collaboration avec Baudouin Dricux, qui m'a par ailleurs permis de confronter les idées universitaires et la réalité industrielle.

Bertrand Meyer, Roger Rousseau et Gérard Comyn ont eu la gentillesse de consacrer une partie de leurs loisirs à lire cette thèse. Leurs nombreuses critiques constructives me confirment dans l'opinion que ce travail est loin d'être terminé.

La recherche n'est pas le fait des seuls chercheurs ; je remercie tous ceux à qui incombent les tâches techniques et administratives, plus ingrates mais tout aussi nécessaires, et en particulier Patricia et Madame Debock, sans qui ce document serait resté à l'état de notes illisibles.

Mais ces deux années ne peuvent en aucun cas se résumer dans ces quelques pages. En quittant Lille, j'ai le sentiment de perdre beaucoup : une certaine qualité de relations humaines qui règne "au labo", malgré les inévitables conflits, et une rare authenticité des relations affectives qui lient le cercle heureusement non fermé de mes amies et amis.

L'ailleurs pourra-t-il être meilleur ?

B. Petitjean

TABLE DES MATIÈRES

PRÉLIMINAIRES

Les capacités de traitement des machines informatiques ont évolué beaucoup plus vite que leurs possibilités de communication avec l'utilisateur. Le "dialogue" homme/machine est encore essentiellement dirigé par le système. Un tel "dialogue" est en effet simple à programmer et il permet de conserver les habitudes acquises par le traitement par lots sans aucune interaction.

Certaines recherches tendent à permettre l'écriture des programmes par l'utilisateur même non informaticien (Query By Example ...). Ceci n'est possible que pour des applications très spécifiques et limitées. Le dialogue ainsi défini n'est pas forcément plus interactif, mais l'utilisateur domine mieux ce que la machine lui demande.

Une communication homme/machine idéale serait le dialogue au sens habituel du terme. L'utilisateur exposerait le problème à résoudre d'une manière plus ou moins informelle, la machine demandant des précisions éventuelles et trouvant seule les données nécessaires et le traitement adéquat. Les progrès de l'intelligence artificielle (reconnaissance d'une langue naturelle ...) ne permettent pas d'envisager un tel scénario, qui restera donc de la science-fiction, tout au moins pour le court et moyen terme.

Entre le monologue machine/homme et l'utopie, il faut donc trouver un compromis.

Cette thèse présente des mécanismes généraux permettant d'offrir une grande liberté de manoeuvre à l'utilisateur sans exiger un travail de programmation trop important. La constatation suivante justifie l'orientation proposée :

Tout programme interactif peut se dérouler suivant deux aspects :
- une exécution "normale", où les différentes phases s'enchaînent suivant un scénario bien établi et facile à programmer.
- des déroutements imprévisibles, provoqués par l'utilisateur.

Ceci concerne, par exemple, un abandon du travail en cours, une demande d'aide ou une reprise sur une phase de l'exécution déjà terminée.
Aucun des langages et outils existants n'offre de moyens systèmatiques pour traiter ce second aspect. Par conséquent, la gestion de l'interactivité, donc des déroutements imprévisibles, se révèle très vite rédhibitoire.

Ce travail proposera donc un mécanisme non explicitement programmé permettant la gestion des reprises de l'exécution ; ce mécanisme implicite sera basé sur la structure générale du programme.

0 - PRÉSENTATION - RÉSUMÉ DE L'ÉTUDE

L'explosion informatique des dernières années a mis de plus en plus d'employés en contact avec le "mystérieux écran vert". Quelques auteurs se sont intéressés à la psychologie et aux besoins de ces utilisateurs non-informaticiens. Ils en ont déduit les qualités attendues d'un bon interface homme/machine.

Le dialogue doit être non ambigu, c'est à dire informer clairement l'utilisateur du "pourquoi" et du "comment" des actions demandées. La précision de l'interface doit éviter tout recours aux documents annexes ou au personnel plus qualifié. Les messages devront donc être judicieux, c'est à dire complets et concis, ce qui est souvent contradictoire. Ceci conduit à proposer plusieurs versions : un dialogue rapide, pour l'utilisateur chevronné, et un dialogue très complet, de type enseignement assisté par ordinateur, l'objet du cours étant l'utilisation de l'application. Dans tous les cas, la disponibilité d'une "aide" doit permettre de lever les ambiguïtés. Cette personnalisation du dialogue pourra être très élaborée si le système permet la sauvegarde d'un profil utilisateur.

Si le dialogue doit s'adapter à l'utilisateur, il devra par contre masquer la diversité des matériels et logiciels utilisés. Ceci est d'autant plus nécessaire que les réseaux hétérogènes se développent.

Mais le meilleur interface du monde ne pourra empêcher l'utilisateur de commettre des erreurs. Il faudra donc vérifier toutes les entrées et en faciliter la correction éventuelle. Les erreurs de syntaxe ne posent aucun problème majeur. Encore faut-il laisser à l'utilisateur une certaine souplesse dans la formulation des entrées. Les erreurs liées à la cohérence sémantique des données sont plus difficiles à définir. Elles concerneront souvent des contraintes mutuelles entre plusieurs entrées.

Enfin, certaines données, même acceptées par le système, pourront ne pas correspondre au désir de l'utilisateur. Un dialogue vraiment souple lui permettra de modifier l'une ou l'autre de ses décisions préalables. L'écriture de dialogues présentant toutes ces caractéristiques est actuellement tout à fait possible mais se révèle très longue et coûteuse. Des outils de génie logiciel ont été proposés pour faciliter cette écriture. Trois niveaux sont alors à envisager :

- la phase d'exécution d'un dialogue particulier pour un utilisateur
- la définition du dialogue par le programmeur
- la spécification et l'implémentation de l'outil de génie logiciel par le concepteur.

Ce seront évidemment la souplesse de l'exécution et la facilité de la programmation qui justifieront l'outil logiciel proposé.

Tous les logiciels (programmes et outils) s'accordent à ne plus traiter les échanges caractère à caractère, ni même ligne à ligne, mais à utiliser les "deux dimensions". Ceci s'adapte parfaitement aux terminaux actuels (clavier/écran), et permet de simuler le travail sur formulaire auquel les employés sont habitués.

Un dialogue par grilles d'écran présente de nombreux avantages pour l'utilisateur : visualisation du contexte, saisies en bloc permettant les corrections internes au bloc, regroupement logique des données ... De plus, la définition d'une grille en tant que module facilite l'écriture et la modification des programmes ; elle permet la séparation structures de données/messages et l'indépendance par rapport au matériel utilisé ...

Suivant les outils logiciels, une grille est une entité plus ou moins complexe. Aux extrêmes, on trouve "SBA" (System for Business Automation), aux notions simples permettant à l'utilisateur de définir lui-même ses applications et un langage de définition de grilles utilisant les concepts du langage ADA (Gehani).

On peut pourtant discerner deux tendances : les outils logiciels qui se limitent à faciliter la définition des grilles, et ceux qui s'intéressent plus à l'enchaînement entre les différentes grilles.

Le déroulement du dialogue est en général défini à l'aide des structures de contrôle habituelles aux langages de programmation. L'outil logiciel GESCRAN innove en permettant la définition des dialogues à l'aide de graphes : un noeud du graphe correspond à une grille et les transitions représentent l'ensemble des possibilités d'enchaînement de ces grilles.

Mais l'ensemble des outils proposés reste tributaire du schéma classique de transaction-formulaire indivisible. Une transaction se déroule en trois phases

affichage de la grille, saisie-vérification-validation, exécution des traitements demandés avec effacement de la grille. Ce schéma n'est plus adapté lorsqu'il s'agit d'applications non transactionnelles. Un logiciel décrit dans [KAISE 82] assouplit les contraintes en retardant le traitement associé : on a alors un traitement par lots (non interactif) avec saisie de données interactives (structuration et vérification des données avant exécution).

Le langage proposé dans ce travail tente d'allier la souplesse du dialogue avec une exécution "temps réel" des traitements associés. Il s'appuie sur la constatation suivante. Tout programme interactif se déroule dans un des deux modes suivants :

- un mode normal, où les différentes phases s'enchaînent suivant un schénario bien établi et facile à programmer
- un mode "déroutement", lorsque l'utilisateur revient sur une phase déjà exécutée.

La possibilité de gérer automatiquement ces déroutements va soulager considérablement l'effort de programmation d'une application.

Les mécanismes proposés vont exiger du programmeur qu'il s'attache à structurer convenablement son application, à définir les structures de données nécessaires et leurs types, et à définir les contraintes sémantiques sur les données. Ces trois points relèvent plus de l'analyse du problème que du "transcodage" dans un langage particulier, ce qui est en accord avec la tendance actuelle des langages de programmation.

La mise en écran, c'est à dire la définition exacte de ce qui sera vu par l'utilisateur, se fera ultérieurement grâce à un éditeur interactif. Le programmeur pourra associer des attributs d'affichage (fenêtres d'écran, messages ..) non seulement aux différentes variables échangées, mais aussi aux blocs de programme.

La structuration en blocs aura un double impact sur l'exécution du dialogue :

- elle réglera les moments d'affichage et d'effacement des "rubriques de saisie" (notion de portée habituelle)
- elle influencera les possibilités de reprise.

Le mécanisme de reprise est simple, du moins vis-à-vis de l'utilisateur : tout ce qui est visible est modifiable, et aucune donnée antérieure à la donnée modifiée ne sera à re-saisir. Par contre, toutes les données postérieures au lieu de reprise sont perdues. Pour que les possibilités de reprise ne soient pas indûment limitées par la taille d'un écran particulier, un écran virtuel, de taille quelconque, sera géré.

L'affichage d'une rubrique de saisie pendant la durée du bloc englobant se révélant parfois insuffisant, le programmeur pourra déclarer un bloc comme "rémanent" : cette déclaration conservera l'affichage (donc les possibilités de reprise) jusqu'à la fermeture d'un bloc englobant non rémanent.

Pour satisfaire ces spécifications, l'implémentation devra régler plusieurs problèmes.

Tout d'abord, la possibilité de reprise entraîne la possibilité d'annuler une partie de l'exécution d'une application. Il faudra donc pouvoir rétablir le contexte tel qu'il était avant l'exécution de cette phase, sans que le programmeur ait à écrire cette restitution.

Ceci sera réalisé par la création de points de reprise (sauvegarde du contexte) liés à la structuration en blocs de l'application. Une reprise à l'intérieur d'un bloc rémanent déjà terminé est possible. La hiérarchie des blocs de reprise sera donc insuffisante, et le système devra gérer une trace du dialogue, ce qui évitera de redemander à l'utilisateur des données antérieures à la donnée qu'il veut modifier.

Enfin, une structure arborescente représentant l'écran virtuel permettra de gérer les défilements "en rouleau" à l'intérieur des fenêtres (sous-écrans) réservées aux blocs et aux structures multiples (tableaux ...).

Une implémentation minimale utilisant des structures linéaires sera proposé. Diverses extensions seront envisagées : possibilité de permettre à l'utilisateur chevronné de modifier la "mise en écran", possibilité de réutiliser les données postérieures à la modification en cas de reprise ... Enfin, une intégration du langage de dialogue dans un environnement ADA sera évoquée.

Pour clore ce travail, une réflexion plus générale sur un modèle d'interaction sera menée, à la lumière de différents travaux réalisés sur ce sujet à l'Université de Lille. La conclusion privilégiera un modèle arborescent, offrant la plupart des possibilités du graphe tout en évitant sa complexité.

1 - QUALITÉS D'UN INTERFACE CONVERSATIONNEL

Avant d'aborder l'étude des outils existants ou à créer, il est bon de cerner les qualités d'un interface que l'utilisateur est en droit d'exiger. Plusieurs ouvrages se sont intéressés à la psychologie et aux besoins de l'utilisateur de programmes interactifs. Le premier en date est sans doute [MARTI 73]. Les recherches dans ce sens sont toujours d'actualité : [DEAN 82], [HAYES 81] ...

On s'accorde à dire que l'utilisateur est et sera de plus en plus non-informaticien. On admet que la puissance de calcul de la machine, déjà bien suffisante pour nombre d'utilisations actuelles, ira grandissant. Ces deux facteurs entrainent que la qualité d'un produit logiciel pourrait presque se résumer à la qualité de l'interface ; tout au moins, l'utilisateur le percevra comme cela. Ceci n'est évidemment pas une critique des nombreuses recherches visant à augmenter les performances des machines ou des programmes. Mais hélas, le décalage croissant entre la sophistication des logiciels et la pauvreté des outils d'écriture de dialogue homme/machine entraîne une sous-utilisation de ces logiciels. Bien que les outils deviennent de plus en plus puissants, il faut que les interfaces se simplifient. A quoi bon mettre un terminal à la disposition de chaque employé d'une entreprise s'il n'est pas capable de se servir des logiciels accessibles. Les caractéristiques exposées ici se rapportent à un dialogue alphanumérique. Ceci se situe bien en retrait des techniques de communication de pointe : éditeur d'images, synthèse et reconnaissance de la parole ... Mais les quelques principes abordés sont très généraux et pourraient sans problèmes être appliqués aux outils de communication plus évolués.

L'utilisateur d'un interface de dialogue en attend quelques qualités.

1.1 - Le dialogue doit être un guide non ambigu : il doit informer à tout moment l'utilisateur sur les données à fournir au programme. Comme toujours, une donnée se caractérise par une sémantique (que demande le programme ?) et une syntaxe (sous quelle forme lui donner ?). Lorsqu'une saisie est acceptée, il est bon que le programme fasse part à l'utilisateur de l'interprétation qu'il en a faite. La demande de mise à jour d'un solde de compte pourra par exemple être l'impression du message :

```
- Mise à jour du compte DUPONT R. -
     Débit ? (ex  2000.35) : |_|
```

La réaction du système pourra être l'impression du message :
compte DUPONT R : Débit de 370 F ; nouveau solde : 4025.80 F
On peut demander à l'utilisateur de confirmer la transaction. La mise à jour effective ne se réalisera alors qu'après la validation.

Un interface bien documenté doit rendre le recours à une notice utilisateur inutile. Le problème du choix des messages est évoqué par Dean [DEAN 82].

1.2 - Choix des messages

La qualité des messages a un grand impact sur l'utilisateur ; Le programmeur doit choisir judicieusement ses messages, c'est à dire fournir des messages complets mais concis. Si trop de notions sont introduites par le même message, l'utilisateur devra le relire plusieurs fois. De bons messages seront compréhensibles par tous (termes concrets et familiers) mais bien spécifiques à l'application (termes exacts et précis). Plusieurs méthodes peuvent être employées pour obtenir des messages courts : compacter les messages (abréviations ...), résumer ou n'évoquer qu'une seule des notions à aborder. Dean sépare trois aspects d'un message : le sens, l'information choisie et le langage utilisé. Par exemple, pour demander aux visiteurs d'un jardin public de ne pas cueillir les fleurs (sens du message), on peut donner comme information : les fleurs sont rares et chères, ou chacun doit pouvoir en profiter, ou la cueillette sera punie d'une amende. L'expression de chacune de ces informations peut se faire sur un ton plus ou moins dissuasif.

La conclusion de l'article est que la définition précise du dialogue doit précéder la définition du programme lui-même. Pour cela, il faut commencer par simuler l'application à programmer, c'est à dire se "mettre en situation". Ce qui est en contradiction avec la méthode habituelle : on écrit le programme en y incorporant quelques messages succincts. On revient ensuite sur ces messages pour les rendre explicites.
Quelle que soit la qualité des messages, une erreur de l'utilisateur est toujours à prévoir.

1.3 - Le dialogue doit être résistant aux erreurs :

Lors d'un traitement par lot, une erreur dans les données d'entrées entraîne l'arrêt de l'exécution du programme. En utilisation interactive, il faut profiter de la présence "en ligne" de l'utilisateur pour corriger les erreurs :

il faut donc les détecter en temps réel. On distinguera les erreurs de syntaxe des erreurs de sémantique, beaucoup plus délicates à déceler.

1.3.1 - Les erreurs de syntaxe : la demande d'une donnée doit indiquer la syntaxe attendue. L'utilisateur entre sa donnée ; un analyseur syntaxique en temps réel doit accepter ou non cette donnée. On peut envisager une analyse à la volée (caractère par caractère) ou se contenter d'une analyse globale en fin de saisie de la donnée. La première semble plus séduisante : blocage du curseur dès qu'un caractère est incorrect, ce qui évite à l'utilisateur de taper toute la donnée. Mais elle risque de dérouter certains utilisateurs. En cas d'erreur de syntaxe, le curseur se repositionne au début du champ de saisie, un message en informe l'utilisateur (en précisant éventuellement la syntaxe exigée) et une nouvelle saisie ou une correction est demandée.

La syntaxe exigée pour les variables des langages de programmation est en général très stricte. Si les sommes utilisées par un programme doivent être arrondies au franc inférieur, les données 100.2 et 100.0 seront refusées. On peut envisager de les accepter, le programme comprenant alors 100. Mais la validation doit rester à la charge de l'utilisateur, puisque 1002 ou 1000 sont également des interprétations possibles.

Pour assouplir la syntaxe, il est possible d'associer à chaque valeur d'un type un voisinage : toute donnée appartenant à un voisinage d'une valeur sera interprétée comme cette valeur.
Le voisinage peut être logique : OUI = 0 = YES = Y
ou physique : touches contigües à une touche donnée ...
Il faut évidemment que les voisinages de deux valeurs différentes soient disjoints (ce qui n'est pas le cas pour les touches).
Dean [DEAN 82] définit plusieurs niveaux de réactions d'un programme par rapport à une donnée :
- donnée immédiatement utilisable (pas d'erreur).
- correction évidente
- correction non certaine
- donnée inutilisable.

En cas d'erreur suivie d'une correction, l'utilisateur doit être consulté pour validation.

En augmentant les performances de l'analyseur syntaxique, on peut donc assouplir considérablement le dialogue. Les erreurs liées à la sémantique des données sont par contre plus difficiles à traiter.

1.3.2 - *Erreurs liées à la sémantique des données*

Une donnée n'est pas isolée : le sens qui lui est attaché dépend d'autres données et du programme qui va l'utiliser. La sémantique d'une donnée provient donc de son environnement. On peut distinguer deux types de cohérence : la cohérence intrinsèque et la cohérence d'ensemble. Cette distinction reflète une différence de liaison sémantique avec l'environnement et va exiger des traitements d'erreur appropriés.

La cohérence intrinsèque d'une donnée fait intervenir le contexte dans lequel le programme va travailler : est-ce que le nom de client qui vient d'être saisi appartient à la liste des clients de l'entreprise ? ... mais en cas de non respect de la règle sémantique, seule la donnée saisie doit être modifiée. A ce stade, l'environnement est considéré comme correct.

La cohérence d'ensemble concerne un groupe de données, liées par une assertion de cohérence d'ensemble ; ce groupe pourra donc être appelé ensemble de cohérence. Par exemple, la relation

"l'ensemble des mouvements sur un compte (crédits et débits) ne doit pas rendre négatif le solde du compte"

crée un ensemble de cohérence comprenant le solde avant transaction et les mouvements de la transaction.

En cas de non-respect de l'assertion, toutes les données saisies de l'ensemble de cohérence sont suspectes. Pour que l'utilisateur puisse effectuer les corrections nécessaires, il faudra donc lui donner accès à des données déjà acceptées : la dernière saisie n'est pas obligatoirement la donnée erronée.

La distinction cohérence intrinsèque d'une donnée et cohérence d'ensemble vient donc de la confiance ou de la méfiance du programmeur vis-à-vis du contexte de cette donnée au moment de sa saisie. En première approche, sont suspectes toutes les données saisies non encore expressément validées ; dans l'exemple des clients d'une entreprise, les noms de clients ont été saisis lors d'autres transactions, et la demande de consultation des commandes d'un client non existant ne remet pas en cause ces données.

Le message à délivrer en cas d'erreur de cohérence est délicat à déterminer. Il pourra être déduit de l'expression de l'assertion de cohérence ou défini explicitement.

Le contenu du message peut d'ailleurs dépendre du type de l'utilisateur en ligne.

1.4 - Le dialogue doit être personnalisé

Il doit être adapté au niveau de l'utilisateur, à son langage et à ses dernières utilisations du programme. Les messages attendus par un utilisateur débutant ne sont pas les mêmes que ceux désirés par un utilisateur confirmé et pressé. Au niveau débutant, le dialogue sera très directif : de type Enseignement Assisté par Ordinateur, le but de l'enseignement étant évidemment l'utilisation du programme. Un utilisateur occasionnel se contentera de messages explicites. Quand à l'utilisateur pressé, il préferera un texte réduit à quelques abréviations n'encombrant pas inutilement son champ de vision. Dans tous les cas, un aide-mémoire disponible permettra de réaliser un compromis entre la concision des messages et la donnée d'un maximum de renseignements : touche spécialisée "aide". L'aide pourra être graduée (plusieurs niveaux) et spécifique à la situation courante (une erreur vient d'être commise ou non ...).

Un programme d'application peut être utilisé par des spécialistes du domaine d'application ou au contraire par des novices. Le cas le plus courant est celui d'utilisateurs non-informaticiens. Le dialogue devra en tenir compte : certaines formulations, triviales pour les premiers resteront incompréhensibles pour les autres.

Le dialogue doit également tenir compte de la langue maternelle de l'utilisateur. On peut ne pas comprendre l'anglais mais vouloir quand même utiliser des programmes.

Enfin, deux utilisations successives d'un programme par la même personne ont souvent beaucoup de points communs. La possibilité de commander "la même chose" peut réduire considérablement les saisies. Un dialogue vraiment personnalisé se souviendra des précédentes "rencontres", et permettra la définition d'abréviations propres.

Si le dialogue doit s'adapter à l'utilisateur, il est par contre préférable qu'il ne dépende pas des outils matériels et logiciels utilisés.

1.5 - Le dialogue doit être uniforme : un terminal peut être connecté sur un réseau hétérogène. Le programme d'application pourra éventuellement être exécuté par des machines différentes (bases de données distribuées ...) Une même application peut effectuer un même travail à l'aide d'algorithmes différents (algorithme mieux adapté à l'ensemble des données saisies ...) Tout ceci doit rester complétement transparent à l'utilisateur.

1.6 - Le dialogue doit être souple et efficace : l'utilisateur doit avoir la possibilité d'abandonner le travail en cours sans dommage : il doit laisser le système dans un état cohérent, (fichiers fermés, enregistrements libérés ...). La possibilité de reprendre le travail au point d'abandon est très séduisante : problème des plages horaires fixes ...

Un dialogue efficace (au sens utilisateur) doit exiger un minimum de saisie et avoir un temps de réponse minimal. On qualifie de réaction en "temps réel" une réaction dans un délai de l'ordre de la seconde, à la rigueur de quelques secondes.

Les saisies pourront être limitées si le programme fournit toutes les données qu'il peut retrouver par lui-même. Ces deux exigences sont hélas souvent contradictoires.

En conclusion, un bon interface conversationnel remplacera une notice utilisateur. Il tolérera les erreurs, qui devront être aussi simples à corriger que difficiles à commettre. Il sera personnalisé et indépendant des matériels utilisés. Il devra être souple et efficace. Ces caractéristiques rendent l'interface fortement dépendant du contexte. Un bon dialogue doit gérer un profil utilisateur, une trace de l'exécution permettant les reprises pour correction, et un profil de l'environnement matériel pour décharger l'utilisateur des problèmes de compatibilité des saisies.

Ayant défini les caractéristiques attendues d'un interface conversationnel, on peut maintenant comparer certaines réalisations actuelles d'outils d'aide à l'écriture d'interface.

2 - OUTILS D'AIDE À L'ÉCRITURE D'INTERFACES CONVERSATIONNELS

Parmi les outils d'aide à l'écriture de dialogue, on peut distinguer deux tendances ;

- les logiciels qui se limitent à l'écriture et à la gestion de grilles d'écran
- les logiciels qui proposent un véritable outil de définition du dialogue dans lequel vont s'intégrer ces grilles.

Quelques exemples de logiciels de gestion de grilles seront étudiés : l'utilitaire DFC sur MINI 6, un utilitaire MINI 6 développé à l'IUT informatique de Lille, le langage INTERROGIP II, des Houillères du Nord. Parmi cette tendance, deux extrêmes sont intéressants à comparer : "System for Business Automation", ne nécessitant pas d'informaticiens, et un langage de très haut niveau proposé par Gehani [GEHAN 83].

D'autres logiciels s'attachent plus à décrire le déroulement du dialogue, c'est à dire l'enchainement des grilles et l'effet des interactions dialogue - utilisateur. On étudiera une extension PASCAL interactif proposée par Lafuente et Gries [LAFUE 78], Screen Rigel, attaché à une base de données, les logiciels GESCRAN et CONSCRAN du laboratoire EDF de Clamart, le langage LOGTRAN développé par la SERLIE, ainsi qu'une étude des concepts de reprise implémentés par ce langage [CARRE 80]

Trois critères guident cette étude :
- facilités de définition et de mise à jour des grilles
- séparation et communication dialogue-traitement
- contrôle des erreurs et possibilités de reprise.

Suivant le type du produit et les sources de documentation, certaines rubriques seront favorisées. Les exemples donnés sont repris textuellement des articles, notices ou thèses utilisés.

2.1 - DISPLAY FORMATTING & CONTROL

(DFC ; produit CII, Notice utilisateur).

DFC est un utilitaire, disponible sur MINI 6, facilitant l'écriture et l'utilisation des grilles d'écran, indépendamment (?) d'un matériel (terminal) particulier. Remplaçant le dialogue "questions-réponses sur télétype" habituel, ce produit vise à réduire l'effort de programmation et à augmenter la vitesse et la fiabilité des saisies, tout en aidant à la standardisation des logiciels.

Le document ayant servi à cette étude étant une notice utilisateur, les problèmes d'implémentation ne sont pas abordés.

2.1.1 - *Description/création/modification des grilles*

Une grille est une suite de champs fixes, variables ou non affichés. Un champ fixe est un titre, une question, précédent un champ de saisie. Un champ variable est un champ de saisie ou d'édition. Un champ non affiché contiendra des données utilisées par le programme d'application (mémoire temporaire, communication interprogramme ...). Des valeurs sont associées à chaque champ et déterminent les numéros de ligne et de colonne, le nom, les attributs, les caractéristiques d'édition, la valeur initiale, les limites, la présence éventuelle d'une extension. Le nom (facultatif) permet au programme d'adresser un champ particulier. Un numéro est donné par défaut (de gauche à droite et de haut en bas). Les attributs spécifient la longueur du champ et le type de chaque caractère de ce champ, ainsi que les insertions automatiques de caractères dans la chaîne de saisie ; par exemple, lorsque l'utilisateur doit entrer une date : CC/CC/CCCC ; C pour Chiffre, les "/" ne sont pas à taper.

L'édition permet :
- de cadrer à gauche ou à droite
- de compléter par des zéros ou des blancs
- de laisser ou non à l'utilisateur la possibilité de ne pas saisir un champ
- de dupliquer un champ sur plusieurs appels de la même grille
- d'interdire le changement de valeur d'un champ après la saisie.

L'extension, qui peut être associée à chaque champ, est utilisable par le programme d'application.

La conception des grilles est interactive : la création et la modification se font par l'intermédiaire de grilles spécifiques. L'écran est alors divisé en trois parties :

- la ligne des messages systèmes
- l'affichage de la grille ou partie de grille en cours de conception
- les spécifications des champs.

2.1.2 - *Interface Programme d'application/dialogue*

Les grilles sont utilisables par l'intermédiaire d'appels de procédures, lorsque le terminal est en mode grille. Les appels (COBOL ou macro-assembleur)

sont divisés en quatre groupes :
- appels d'initialisation : connecte un terminal, lui associe les grilles et "remet à zéro" l'écran
- appels de mouvement de données : échange (lecture ou écriture) des données entre les tampons utilisateurs, le tampon du terminal et l'écran
- appels de fin de session : déconnecte le terminal
- appels de communication avec l'utilisateur : envoi d'alarme, échange de messages sur la ligne d'écran réservée au système.

2.1.3 - *Contrôles de saisie et reprises*

La définition de chaque caractère de saisie (numérique, alphabétique ou alphanumérique) permet un contrôle "à la volée". Les contrôles de respect des bornes et de saisie obligatoire sont effectués lors des appels pour lecture de grille. On a donc un contrôle syntaxique en temps réel ou en très léger différé mais de bas niveau.

Les reprises sont simples sur la grille courante (commande de re-lecture) et toujours possibles sur une grille quelconque par programme.

DFC facilite donc l'écriture de dialogue en permettant au programme de gérer des grilles d'écrans. Chaque grille est déterminée par la donnée de ses champs, ce qui soulage de la définition caractère par caractère.

S'inspirant de DFC, un utilitaire à été développé au département Informatique de l'IUT de Lille. Dans certains cas précis, il permet la génération automatique des programmes d'utilisation des grilles.

2.2 - LOGICIEL de GESTION de GRILLES d'ECRAN - IUT Avril 82 - Duthilleul - Grimonprez

Notice Utilisateur.
Ce logiciel, inspiré de DFC, a pour objectifs :
- une définition aisée et une maintenance simple des grilles d'écrans
- une aide à la programmation : génération possible de programmes COBOL de création d'un fichier séquentiel (saisie) et/ou de mise à jour d'un fichier séquentiel indexé.

Le logiciel est opérationnel sur MINI 6, l'ensemble des sous-programmes de gestion de grilles étant appelable en COBOL.

2.2.1 - *Structure - Création - Modification d'une grille*

Chaque description de grille est un fichier source de 24 lignes. Les créations et modifications se font par l'éditeur de texte habituel. Les zones à saisir sont de 3 types : numérique, alphabétique ou alphanumérique.

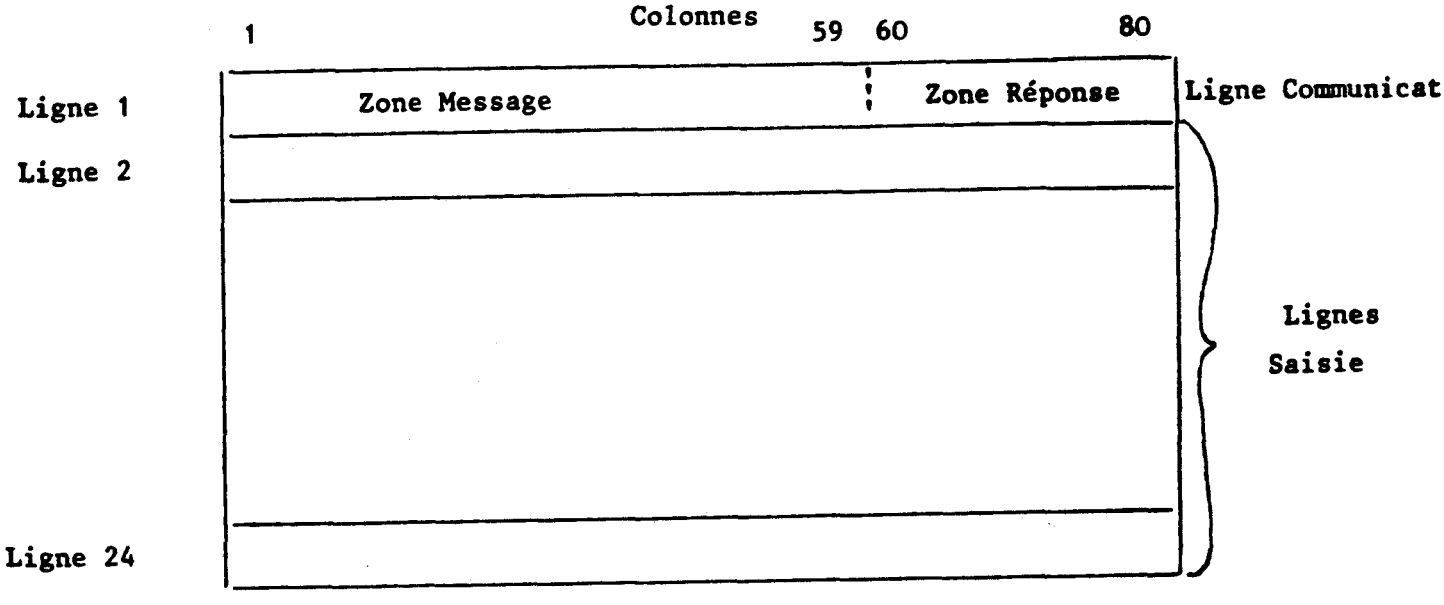
2.2.2 - *Utilisation*

Un programme COBOL peut utiliser un nombre quelconque de grilles : il lui suffit de les déclarer.
Les sous-programmes d'exploitation de grille permettent la remise à zéro et les transferts de données. Ceci est donc très proche de DFC. Mais des utilitaires spécifiques simplifient le travail de programmation. Ils permettent de générer les programmes de création et de mise à jour des fichiers correspondant aux grilles ; cela est possible parce que l'utilisation principale des grilles est la saisie/modification d'articles de fichier. Dans ce cas simple d'utilisation, le travail est donc réduit au minimum.

2.2.3 - *Exemples*

1) Structure d'une grille d'écran

1) STRUCTURE D'UNE GRILLE D'ECRAN



Exemple de fichier source définissant une grille :

```
umero de dossier :!x!!XXYXX!   Numero INSEE (3 premiers chiffres) :!999!
om :!XXXXXXXXXXXXXXXXXXXXXXXX!Prenom :!XXXXXXXXXXXXXXX! Code postal :!99999!

ec.(OUI ou NON):!XXX! Serie :!Xx!   Mention :!XX!    Annee :!99!
tablissement nom :!XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX!  Code postal :!99999!

LASSE DE PREMIERE              | CLASSE DE TERMINALE
redouble :!XXX!                |  redouble :!XXX!
         1       2       3     |                1       2       3
Fran.  !99!    !99!    !99!    | Philo.    !99!    !99!    !99!
Lang.  !99!    !99!    !99!    | Lanc.     !99!    !99!    !99!
Math.  !99!    !99!    !99!    | Math.     !99!    !99!    !99!
Phys.  !99!    !99!    !99!    | Phys.     !99!    !99!    !99!
                               | eco.      !99!    !99!    !99!

NSEIGNEMENT SUPERIEUR (OUI ou NON):!XXX!
lasse prepa.(OUI ou NON) :!XXX!      lieu :!XXXXXXXXXXXXXXXXXXXXXXXXX!
eculte (OUI ou NON) :!XXX!           lieu :!XXXXXXXXXXXXXXXXXXXXXXXXX!
nombre d'annee :!9!       diplome prepare :!XXXXXXXXXXXXXXX!
                          diplome obtenu  :!XXXXXXXXXXXXXXX!
```

2) Représentation interne (Mémoire Centrale) d'une grille

La structure suivante permettant de définir une grille dans un programme COBOL :

```
01 GRILLE.
   02 NOM PIC X(12).
   02 FILLER PIC X(2944).
```

est implémenté sous la forme :

```
01  GRILLE.
    02  NOM PIC X(12).                      nom de la grille.
    02  ECRAN ØCCURS 24.                    image de l'écran.
        03  TEC OCCURS 80 PIC X.
    02  N COMP-1.                           nombre de champs à saisir.
    02  FILLER PIC X(6).                    réservé pour extensions.
    02  T OCCURS 127.
        03  LONG COMP-1.                    longueur du champ.
        03  LIGNE COMP-1.                   n° ligne du champ [1,24].
        03  COLONNE COMP-1.                 n° colonne du champ [1,80].
        03  TYP PIC X.                      type du champ : 9, X ou A.
        03  FILLER PIC X.                   réservé pour extensions.
```

DFC et ses variantes montrent donc que l'on peut réduire considérablement le travail du programmeur. Ceci est d'autant plus vrai que le type de dialogue est bien précisé et spécifique.

Un logiciel d'aide à l'écriture de dialogue a également été développé aux Houillères du Nord, dans un contexte bien particulier : celui d'une gestion en temps réel d'une base de données.

2.3 - INTERROGIP II (Documentation : Thèse de M. Oudin "Conception et implémentation d'applications interactives en contexte industriel") [OUDIN 80]

Contexte : Cette thèse est la description d'un système de gestion de base de données en temps réel sur IBM 370, et du langage qui lui est associé (Interrogip I
Le langage traitera donc des transactions : consultation et mise à jour.

Le dialogue système/utilisateur se fait par l'intermédiaire de grilles d'écran. Une transaction élémentaire (physique) est associée à une grille.

2.3.1 - *Création et modification des grilles*

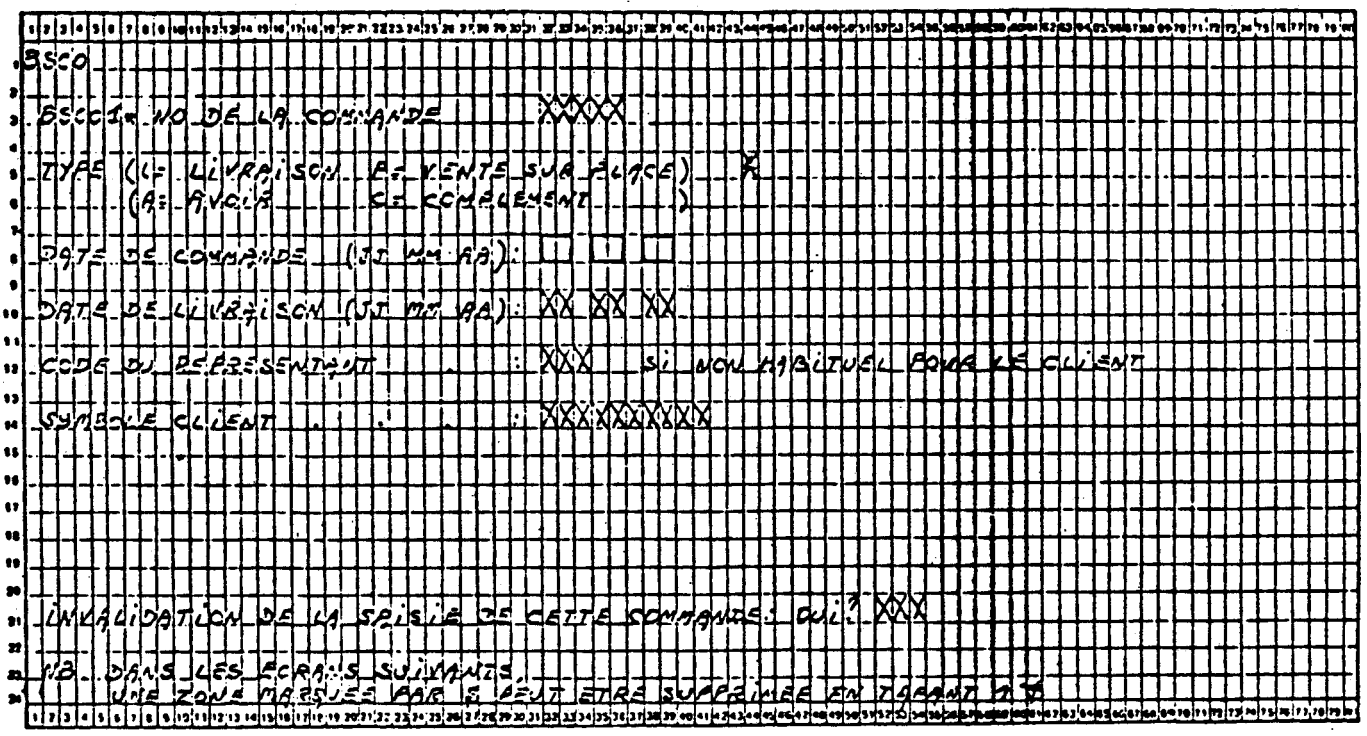
Une grille est composée de champs (zones) avec attributs. Chaque zone est protégée ou non, normale, clignotante ou invisible, à renvoyer ou non au processeur central.

Le programmeur définit sa grille en déclarant ses différentes zones. La création n'est donc pas interactive, il n'y a pas d'"éditeur de grilles".

Le langage permet les boucles dans la définition des grilles. Il propose des facilités pour gérer les écrans multiples (grilles ne tenant pas sur un écran) : l'instruction TITRE permet d'éditer un titre en haut de chaque écran sans comptage de ligne. Le passage d'un écran au suivant se fait par la touche "suite" sans possibilité de retour.

A l'inverse, on peut définir des masques multiples : un même écran résulte alors de l'affichage simultané de plusieurs masques. De plus, des instructions spécifiques permettent de définir certains contrôles.

Exemple :



Ce masque permet à l'opérateur de rentrer les renseignements suivants :

- numéro de la commande
- la date de livraison souhaitée
- le code du représentant, si celui-ci n'est pas le représentant habituel et le symbole du client - il s'agit d'un code mnémonique attribué à chaque client. Ce code est généralement connu ou imaginable par l'opérateur à partir de quelques règles simples.

La date de commande est directement fournie par le programme - c'est la date du jour.

L'opérateur peut aussi annuler une commande déjà enregistrée au moyen de cette grille. Il suffit de répondre OUI à la question correspondante.

La première grille étant remplie, l'opérateur "entre" les données et reçoit un 2ème masque.

2.3.2 - Contrôle de saisie

Une zone à compléter peut être soit numérique, soit alphanumérique. Ceci détermine un contrôle syntaxique local. Les zones protégées sont inaccessibles par le curseur. Les attributs des données élémentaires (déclaration de type, de longueur ...) ne semblent pas utilisés pour le contrôle.

Quelques contrôles de vraisemblance sont prévus :

- si <nom de donnée> est {PRESENTE / ABSENTE} aller à <label>
- test de date
- test de validité par consultation de table ou de fichier.

Des contrôles de cohérence d'ensemble sont possibles mais non facilités. Il suffit de lire les données d'une grille, d'exécuter les instructions nécessaires et de renvoyer la grille (ou une autre) en cas d'erreur.

Une instruction spécifique d'erreur
ERREUR <'texte du message'> sur le CHAMP <nom du champ saisi> et FORCER <nom de transaction>
permet d'afficher un texte en première ligne, de déplacer le curseur sur la zone à re-saisir et de lancer une transaction spécifique.

En plus des contrôles automatiques, les libellés explicites correspondant aux codes saisis peuvent être affichés pour permettre la vérification par l'utilisateur.

Un minimum de contrôle est donc permis. Mais, bien que les descriptions de masques soient en fichier, il semble difficile de séparer une grille du programme qui l'utilise.

2.3.3 - Communication saisie/traitement

Chaque terminal comporte en mémoire une zone de travail accessible par l'utilisateur et une zone de saisie-affichage qui contient ce qui a été frappé au clavier et les textes à afficher.

Une transaction logique est une suite de transactions physiques. Chaque transaction physique possède des données propres contenues dans sa mémoire de

travail. Les données définies sur l'ensemble de la transaction logique sont dans la zone liée au terminal. Ceci a permis d'adopter la solution pseudo-conversationnelle : l'envoi et la réception d'une grille se font par deux transactions physiques différentes (pas d'occupation mémoire pendant que l'utilisateur entre ses données).

L'envoi d'un masque consiste en trois phases :
- écriture d'une grille dans la zone d'E/S du terminal (acquise dynamiquement par programme), éventuellement complétée par des données de la base
- initialisation de la transaction suivante, qui traitera la réponse
- arrêt de la transaction.

Pour la réception de masque, il faut associer les adresses écrans aux noms symboliques utilisés (table construite à la compilation).

Tout ceci concerne donc plus la liaison terminal/ordinateur que la communication dialogue/traitement, puique ces deux processus sont mêlés.

2.3.4 - *Reprises en cas d'erreur*

En cas d'erreur détectée sur la grille courante, on retrouve les données de la transaction physique en cours dans sa zone de travail. Les données déjà validées de la transaction logique sont dans la zone de communication. Elles ne seront pas affectées par la reprise. Ceci donne une certaine structuration en blocs de reprise, (sans possibilité d'imbrication). Cette notion de bloc de reprise sera précisée ultérieurement.

Pour les saisies en masse, il existe un fichier de saisie : cette trace du dialogue est simplement une extension de la zone de communication mais n'est pas utilisée pour les reprises.

Les transactions peuvent s'effectuer sur des doubles de la base, non pour faciliter les reprises "utilisateur", mais pour permettre les reprises en cas de panne.

Les reprises sur des grilles déjà validées ne sont donc pas prévues a priori. Elles restent évidemment possible si la transaction inverse (ou correctrice) peut être lancée par l'utilisateur.

Interrogip II utilise donc des grilles d'écran. Les possibilités de contrôle et de reprise sont peu élaborées. L'intérêt (et le handicap) de ce langage est de s'être développé en fonction d'une base de données spécifique, gérée en temps réel.

Les trois exemples cités (DFC, logiciel IUT, Interrogip II) illustrent bien la notion de grille d'écran : outil de haut niveau facilitant la saisie ou l'édition d'un groupe de données. Mais la gestion des interactions dialogue-utilisateur doit encore être explicite: programmation des reprises ...

La simplicité de la notion de grille d'écran statique a permis l'élaboratio d'un système d'écriture des applications par les utilisateurs non-informaticiens : "System for Business Automation".

2.4 - SBA : System for Business Automation Zloof, De Jong C.ACM 20-6

SBA est un système permettant aux non-informaticiens de "programmer" et d'exécuter leurs applications. L'utilisateur manipule des objets en deux dimensions : tables, formulaires, rapports ... Il peut définir l'application d'une manière incrémentale ; il décrit chacune des phases comme s'il la réalisait manuellement. SBA intégre trois sections :

- un langage de manipulation de base de données (QBE : Query-by-Example)
- un langage de description des applications
- un langage de description des interrelations entre applications.

2.4.1 - QBE : Query-By-Example :

La base de données est organisée suivant le modèle relationnel (Codd 70). Pour décrire une transaction, l'utilisateur va écrire dans un "formulaire" soit des constantes, soit des variables permettant la liaison entre formulaires. Le système affichera les "p-uples" de la base satisfaisant aux relations ainsi précisées (requête "associative").

Exemple : (la Lettre P pour "print", les mots soulignés sont des variables).
Base de données :

EMP

NAME	SAL	MGR	DEPT
JONES	8K	SMITH	HOUSEHOLD
ANDERSON	6K	MURPHY	TOY
MORGAN	10K	LEE	COSMETICS
LEWIS	12K	LONG	STATIONERY
NELSON	6K	MURPHY	TOY
HOFFMAN	16K	MORGAN	COSMETICS
LONG	7K	MORGAN	COSMETICS
MURPHY	8K	SMITH	HOUSEHOLD
SMITH	12K	HOFFMAN	STATIONERY
HENRY	9K	SMITH	TOY

TYPE

ITEM	COLOR	SIZE
DISH	WHITE	M
PENCIL	RED	L
LIPSTICK	RED	L
PERFUME	WHITE	L
PEN	GREEN	S
PENCIL	BLUE	M
INK	GREEN	L
INK	BLUE	S
PENCIL	BLUE	L

SALES

DEPARTMENT	ITEM
STATIONERY	DISH
HOUSEHOLD	PEN
STATIONERY	PENCIL
COSMETICS	LIPSTICK
TOY	PEN
TOY	PENCIL
TOY	INK
COSMETICS	PERFUME
STATIONERY	INK
HOUSEHOLD	DISH
STATIONERY	PEN
HARDWARE	INK

SUPPLY

ITEM	SUPPLIER
PEN	PARKER
PENCIL	BIC
INK	PARKER
PERFUME	REVLON
INK	BIC
DISH	DUPONT
LIPSTICK	REVLON
DISH	BIC
PEN	REVLON
PENCIL	PARKER

Transactions : Requêtes Réponses

1)

TYPE		
ITEM	COLOR	SIZE
P PEN	RED	

ITEM
LIPSTICK
PENCIL

2)

SALES	
DEPT	ITEM
P TOY	BOD

SUPPLY	
ITEM	SUPPLIER
BOD	PARKER

DEPT
HOUSEHOLD
TOY
STATIONERY
HARDWARE

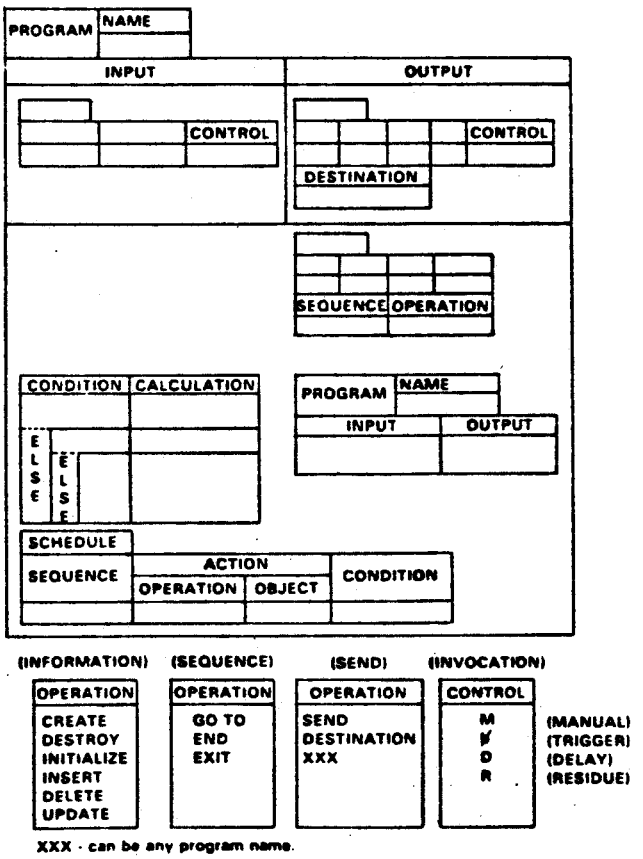
L'utilisateur dispose également de la somme, de la recherche exhaustive (ALL) ...

2.4.2 - SBA : *Construction de programmes*

L'utilisateur peut écrire ses "programmes" en utilisant une syntaxe du même type.

Un programme a un NOM, des ENTREES, des SORTIES et un CORPS.
On retrouve certaines notions classiques : sous-programmes, portées des noms, instructions conditionnelles... On distingue les structures de travail des "structures maîtresses", structures globales incluses dans la base de données. Le déclenchement d'un programme est réglé par des conditions définies par les "ENTREES" (heure, nombre de paramètres minimum ...)

La syntaxe est résumée par le tableau suivant :



Un programme est donc une relation entre des entrées, des objets existants ou à créer et des actions sur ces objets.

2.4.3 - *Conclusion*

En proposant des structures et des opérations très orientées-utilisateur, SBA permet à celui-ci de définir son application d'une manière proche du travail de bureau classique. L'utilisateur peut d'ailleurs ne pas définir complètement toutes les phases de son application. Le système proposé est très unifié : les manipulations sur la base de données et les programmes sont similaires. Il n'y a plus de dichotomie entre la création et l'exécution d'un programme. Mais l'exigence de simplicité entraîne forcément un outil très limité. Ce système s'applique parfaitement aux travaux routiniers et simples à décrire, mais ne peut résoudre les problèmes de définition d'applications complexes.

A l'opposé de cette simplification maximale de la notion de grille, Gehani [GEHAN 83] propose un langage de définition de grilles de très haut niveau.

2.5 - Langage de haut niveau pour la définition de grilles d'écran NH Gehani Computer Journal 26.1.

2.5.1 - *Introduction*

Les avantages de l'utilisation de grilles d'écran en burotique ne sont plus à démontrer :

- une grille regroupe les entités qui sont en relation logique
- les employés sont habitués aux "grilles papier"
- le traitement automatique des grilles est simple (affichage ...)
- les droits d'accès aux grilles et aux champs peuvent être limités.

Les grilles d'écran possèdent donc tous les avantages du "traitement sur papier" en y ajoutant les possibilités de l'informatique interactive.
Un langage de haut niveau pour la définition de ces grilles permettra

- de faciliter la définition, la re-lecture et la modification des grilles
- de regrouper toutes les informations concernant une grille
- de rendre portable les programmes utilisant des grilles
- d'augmenter l'efficacité de chaque implémentation par l'usage d'un traducteur performant.

Le langage proposé par Gehani considère une grille en tant que type abstrait,

c'est à dire un ensemble de valeurs et les opérations qui peuvent y être appliquées. Cette définition de grilles peut être utilisée dans deux contextes différents : un système de manipulation de grilles ou un langage de programmation. Dans cette deuxième approche, plus intégrée, les grilles sont des objets manipulables par le langage au même titre que les autres types structurés.

2.5.2 - *Spécifications du langage de définition de grilles*

Ce langage doit permettre la définition :
- du texte informatif, sans tenir compte d'un type particulier de terminal
- du type et des attributs de chaque champ
- des opérations possibles sur la grille
- de droits d'accès à la grille, aux différents champs et aux opérations
- de pré- et post-conditions, à vérifier avant de commencer la saisie puis en fin de saisie
- de pré- et post-actions, conditionnées par les pré- et post-conditions
- de spécifications des références externes (noms et procédures ...)
- des données et procédures locales
- du traitement des grilles (langage de niveau supérieur)

Le langage SBA (System for Business Automation) propose des facilités de définition de grille. Mais il manque les notions de droits d'accès, la séparation en pré- et post-conditions et actions, des types de champs très utiles, le moyen de définir de nouvelles opérations ...
Gehani propose donc un langage répondant à toutes les spécifications.

2.5.3 - *Mécanisme de définition de grilles*

La syntaxe est inspirée du langage ADA.

Notation :

```
form nom de grille is
     imports ...
     display ...
     fields ...
     operations ...
     access rights
                  fields ...
                  operations ...
     local data and routines ...
end  nom  de grille
```

Exemple : Cette grille permet à un employé de demander une avance pour voyage d'affaires.

La demande sera ensuite signée par le responsable puis envoyée à la comptabilité, qui vérifiera les montants.

grille :

Cash Advance and Ticket Requisition Form # :

Last Name, Initials: Date:
Company-Id # : Room Number: Extension:
Organization # : Case # :

	Employee Only	*Treasury Only*
Cash:	:	
Tickets:	:	
Total:	:	

Funds to be used for:

Will be required until:

Total Amount (in words):

Signatures—Employee:

Supervisor: Date:

Treasury: Date:

définition :

```
form CashAdvance is
  imports

    with EMPLOYEE;
    use EMPLOYEE;

  display
              Cash Advance and Ticket Requisition Form # : $No
     Last Name, Initials: $Name                    Date: $Date_Emp
     Company-Id # : $Id   Room Number: $Room   Extension: $Ext
     Organization # : $Org      Case # : $Case


          Employee Only          Treasury Only
       Cash: $Cash             : $Treas_Cash
    Tickets: $Ticket_Price     : $Treas_Ticket_Price
      Total: $Total            : $Treas_Total

  Funds to be used for: $Purpose

  Will be required until: $Until

  Total Amount (in words): $Amount

  Signatures—Employee: $Emp_Sig

              Supervisor: $Sup_Sig     Date: $Date_Sup

              Treasury: $Treas_Sig     Date: $Date_Treas

  fields
      $No: INTEGER virtual NEXT();
      $Name: STRING(1..30) required;
      $Date_Emp: $Date required;
      $Id: INTEGER range 0..99999 required post{$Name = EMP_NAME($Id)};
      $Room: STRING(1..5) virtual EMP_ROOM($Id);
      $Ext: STRING(1..4) virtual EMP_EXT($Id);
      $Org: STRING(1..4) virtual EMP_ORG($Id);
      $Case: STRING(1..10) required;
      $Cash: FLOAT required post{$Cash > = 0.0};
      $Treas_Cash: FLOAT required post{$Treas_Cash > = 0.0};
      $Ticket_Price: FLOAT required post{$Ticket_Price > = 0.0};
      $Treas_Ticket_Price: FLOAT required post{$Treas_Ticket_Price > = 0.0};
      $Total: FLOAT virtual $Cash + $Ticket_Price post{$Total < = 1500.0};
      $Treas_Total: FLOAT virtual $Treas_Cash + $Treas_Ticket_Price
                              post{$Treas_Total = $Total};
      $Purpose: STRING(1..100) required;
      $Until: DATE required;
      $Amount: STRING(1..100) required;
      $Emp_Sig: SIGNATURE(1..6) required, lock all above
                post{VALID_SIGNATURE($Emp_Sig)};
      $Sup_Sig: STRING(1..100) required, after $Emp_Sig
                post{VALID_SIGNATURE($Sup_Sig)};
      $Date_Sup: DATE after $Sup_Sig;
      $Treas_Sig: STRING(1..100) required post{VALID_SIGNATURE($Treas_
      $Date_Treas: DATE after $Treas_Sig;
  operations
    — — no additional form operations specified
  access rights
    fields
      case DESIGNATION() is
        when TECHNICAL = > update all except $Treas_Cash, $Treas_Ticket_P
                                $Sig_Sup, $Date_Sup, $Treas_Sig, $Date_Trea
        when SUPERVISOR = > update $Sig_Sup, $Date_Sup;
        when TREASURER = > update $Treas_Cash, $Treas_Ticket_Price,
                                $Sig_Treas, $Date_Treas;
      end case;
    operations
      case DESIGNATION() is
        when TECHNICAL | SUPERVISOR | TREASURER = >
                                all except destroy;
        when others = > display;
      end case;
  local data and routines

    N: INTEGER := 0;
    function NEXT return INTEGER is
        N := N + 1;
        return N;
    end NEXT;
end CashAdvance;
```

2.5.4 - *Conclusion*

Le langage proposé, en utilisant la notion de type abstrait et une syntaxe proche du langage ADA, facilite la définition et la modification des grilles d'écran. C'est dans doute la tentative la plus élaborée parmi les différents logiciels étudiés. La mise en écran est à la charge d'une heuristique qui n'est hélas pas précisée. Celle-ci dépendra évidemment du type de terminal utilisé. L'enchaînement entre les grilles n'est pas abordé : si ces possibilités sont intégrées à un langage de haut niveau (ADA par exemple), la programmation de l'application définira le déroulement du dialogue. Comme le souligne l'auteur, certains problèmes restent posés ; par exemple, comment lier des champs de grilles différentes ou comment déverrouiller certains champs pour modification ?

D'autres projets ou réalisations se sont préoccupés de la gestion du déroulement du dialogue. Un exemple est l'extension au langage PASCAL proposée par Lafuente et Gries [LAFUE 78].

2.6 - Extensions du langage PASCAL en vue de faciliter la programmation des dialogues homme/machine

Source : article paru dans IBM J. of RES. & DEV. Mars 78. J.M. Lafuente D. Gries.

Cet article présente un ensemble d'extensions au langage PASCAL permettant l'écriture de dialogues sous la forme de séquences de grilles et de règles de comportement s'appliquant sur ces grilles. Une application interactive est considérée comme une suite de grilles. Chaque grille ("frame") correspond à un état du système : des données sont affichées, l'utilisateur répond aux questions. Ces réponses sont testées (consistance, validité) puis sont mémorisées. Le système traite ensuite ces données pour permettre la transition vers une nouvelle grille (un nouvel état). Chaque état du système correspond donc à une attente de données (système "passif"), le traitement propement dit consistant,entre-autres, à choisir la transition (système "actif").

La description d'une grille doit être indépendante du matériel : le langage permettra de spécifier les entités logiques (rubriques - "items") apparaissant sur l'écran, et le groupement de ces rubriques en sous-grilles.

Les règles de comportement seront spécifiées d'une manière non procédurale : le programmeur n'a pas à définir leur ordre d'exécution.

2.6.1 - *Description d'une grille*

La définition d'une grille peut contenir des déclarations de variables locales, sous-grilles, fonctions et procédures et des rubriques. Une rubrique est une variable à laquelle s'ajoutent des données d'affichage et de saisie :

```
var name:key-in char at (0,6) text 'NAME'
        format A20 ;
```

Une rubrique peut être associée à un fichier. C'est alors une "fenêtre" à travers laquelle on peut accéder à l'enregistrement courant.

Une rubrique peut être en saisie (key-in), en édition (display) ou de type menu (choix entre plusieurs valeurs d'une variable de type ensemble). Une variante du type menu ("attention") permet une réponse immédiate du système lors de la sélection de touches prédéfinies ou de l'utilisation d'un photostyle.

Une grille est un ensemble de rubriques (et de sous-grilles) définie comme unité logique. Elle peut être réutilisée dans plusieurs grilles. L'emplacement de chaque rubrique est relatif à l'emplacement de la sous-grille.

Une grille peut contenir une procédure d'initialisation et une procédure de terminaison.

Des règles de comportement guident l'interaction utilisateur/système. Elles sont de trois types :
- règles d'assignation : forcent la valeur d'une variable suivant certaines conditions
- règles d'obligation : déterminent les conditions nécessaires pour pouvoir terminer une grille
- règles de terminaison : déterminent la fin de la grille.

Les conditions ci-dessus utilisent souvent les trois propriétés booléennes que possède chaque variable de rubrique : assignée ou non, modifiable ou non, affichée ou non.

Il n'est pas prévu d'outil interactif de définition de grilles : le programmeur les spécifie dans le programme d'application lui-même. Le programme de dialogue mêle donc étroitement la saisie et le traitement des données.

Exemple de grille :

```
                BANK OF NEW YORK

                  NEW ACCOUNT

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

STATUS:  *MARRIED
         *SINGLE
```

```
frame F2;
  var addr1: key-in char at (0,19)
             text 'PERMANENT ADDRESS:';
      d3: display text 'MAILING ADDRESS -';
      m1: menu set of (YES. NO) at (0,16)
          text 'Same as above?';
      addr2: key-in char at (0,15) text
             'ENTER ADDRESS:';
  contains title at (0,17); heading; name at (5,3);
           addr1 at (6,3); d3 at (7,3); m1 at (7,21);
           addr2 at (8,6); status at (10,3);
  rules require name;
        require at least 1 of addr1, addr2;
        require addr1 if YES in m1;
        require addr2 if NO in m1;
        with m1: allow only 1 options;
        exclude addr2 if YES in m1;
        with status: allow only 1 options;
        terminate if ENT_KEY
  end;
  procedure initialize;
  begin reset_ frame; ml := [YES]
  end
endframe
```

Exemple de programme :

```
• Program
program bank;
  var title: display text 'BANK OF NEW YORK';
  R: subframe
       return: attention (RETURN);
       rules terminate if return.s end
     end;
  a1: attention (OPEN ACCOUNT, DEPOSIT, WITHDRAWAL)
      at (1,8) text 'Select transaction desired:';
  d1: display text 'NEW ACCOUNT';
  heading: subframe
             inf: display at (4,3)
                  text 'Enter information. Hit ENTER
                  when done.';
             contains d1 at (2,19);
           end;
  name: key-in char at (0,6) text 'NAME:';
  status: menu set of (MARRIED, SINGLE) at (0,9)
          text 'STATUS:';
  accno: display at (0,24)
          text 'The new account No. is:';
frame F1; · · · endframe;
frame F2; · · · endframe;
          · · ·
begin
  start: F1;  • • •
```

2.6.2 - *Communication dialogue/Programme de traitement*

La définition des grilles utilise une extension du langage PASCAL, et il n'y a pas vraiment de séparation dialogue-traitement. Les grilles peuvent être considérées comme des procédures sans paramètre. Un programme comprendra la déclaration de grilles et de procédures, les instructions appelant ces grilles et déterminant leur ordre de présentation à l'utilisateur, ainsi que les instructions de traitement des données saisies.

L'utilisateur n'a pas à se soucier de la notion de grille . Par le jeu des règles de comportement, l'aspect d'une grille peut être totalement modifié en cours de saisie. L'utilisateur peut donc penser passer d'une grille à une autre alors qu'il reste toujours sur la même.

2.6.3 - *Erreurs - Reprises*

La spécification de rubrique format permet d'afficher un message d'erreur lorsque la syntaxe de la donnée saisie est incorrecte. Les variables pouvant être de n'importe quel type PASCAL, la vérification syntaxique est donc assez évoluée.

Les règles de comportement permettent une vérification plus "sémantique" (données obligatoires ...). Elles permettent de lier différentes saisies entre elles.

Leur spécification non-procédurale pose le problème de leur consistance, résolu par l'analyse d'un graphe de dépendance pour déterminer les règles inutiles ou contradictoires.

Mais l'essentiel du travail de prise en compte des erreurs et des reprises reste encore à la charge du programmeur.

2.7 - Programming Language Constructs for Screen Definition Rowe et Shoens

IEEE Transactions on Software Eng. 9.1.

Screen Rigel est un langage de définition de grilles d'écran, facilitant l'écriture de programmes Rigel, langage de haut niveau de manipulation de base de données. Screen Rigel, en utilisant le dictionnaire de description des données de la base, renforce l'indépendance entre programme et données.

2.7.1 - *Introduction*

Les instructions d'entrée-sortie et de gestion de l'interaction occupent 40 à 60 % du texte source des programmes de gestion. Le programme doit en effet s'occuper du formattage des données, de leur indition, du contrôle d'erreur et de la gestion du curseur et de l'écran. Pour alléger cet effort de programmation, Screen Rigel donne des outils de haut niveau de construction de grilles : description des données, du cadre de grille et d'actions spécifiques. L'accès au dictionnaire des données de la base permet de changer le type des données manipulées sans modifier les grilles.

2.7.2 - *Définition des grilles*

Une définition de grille contient trois parties : la description des données, la description du cadre de grille associé (visualisation sur écran) et la description de traitements spécifiques. La déclaration des différents types des données de la grille entraîne la déclaration d'un enregistrement qui permettra la manipulation des valeurs échangées par le programme. Le nom de chaque champ constitue le message associé par défaut. Les données échangées peuvent l'être en édition ("output"), en saisie ("input") ou en lecture pour mise à jour ("modify"). A chaque champ est associé un contrôle syntaxique et éventuellement une procédure : celle-ci permet outre un contrôle plus élaboré, une définition de grilles interactives. Les champs peuvent être initialisés, et le programme travaillant sur des copies, cette initialisation est sauvegardée d'un appel sur l'autre. La déclaration explicite des différents champs n'est pas obligatoire grâce à l'utilisation du dictionnaire des données.

2.7.3 - *Dictionnaire des données*

L'ensemble des programmes manipulant la base de données peut accéder au dictionnaire des données. Celui-ci contient, entre autres, la description des objets de la base : déclarations des types, relations entre ces objets ... Ce centralisme facilite la mise à jour de la structure de la base. Lors de l'ajout d'un champ à un enregistrement, par exemple, il suffit de recompiler les programmes manipulant ce type d'enregistrement.

Screen Rigel utilise cette facilité pour rendre les grilles indépendantes des données, et donc cohérentes avec la structure de la base tout au long de son évolution.

Pour profiter de cette souplesse de programmation, il suffit d'utiliser les options par défaut : un générateur de format construit le cadre de grille

en fonction du type de l'enregistrement (nombre de champs, nombre de caractères par champ) ...

L'utilisation des options par défaut pouvant présenter certains inconvénients (une méthode générale ne permet pas de prendre en compte tous les cas particuliers), le "formattage" peut être écrit directement dans la définition de type du dictionnaire.

Exemples :

Utilisation d'un type défini par ailleurs :

```
person: record
   name: array 1..20 of char;
   age: 1..100;
   salary: real;
   payStatus: (exempt, nonexempt, contract);
   occupation: array 1..10 of char;
   maritalStatus: (single, married, other);
   ssn#: array 1..9 of char;
end;
```

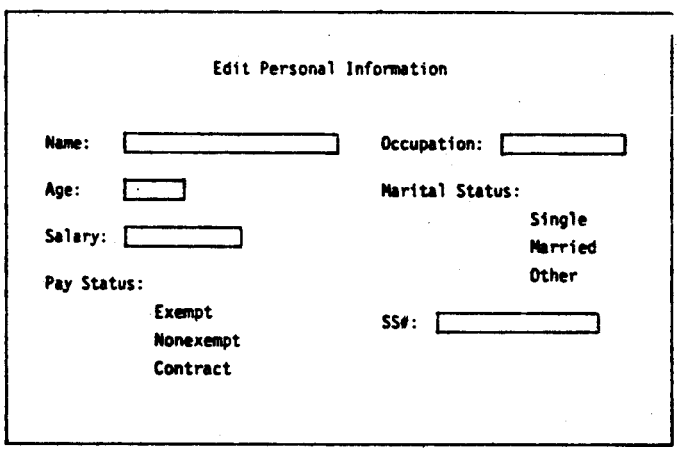
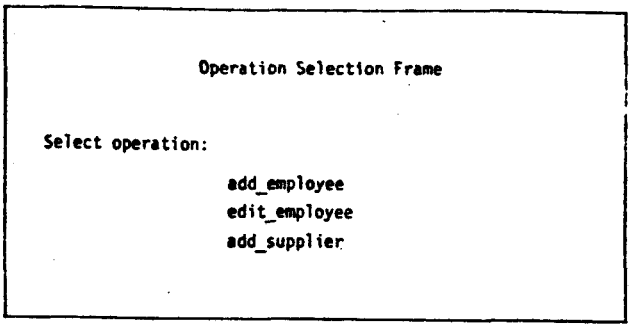


Fig. 3. Sample personal frame.

```
editPerson: frame
    PERSON'record_type;
format
    center("Edit Personal Information");
    column(2,
        output(name);
        modify(age, format: "DDD", check: ageCheck);
        modify(salary, format: "5D.2D", check: salaryCheck);
        modify(payStatus, label: "pay status:");
        modify(occupation);
        modify(maritalStatus, label: "marital status");
        output(ssn#, format: "3D-2D-4D");
    )
end
```

Fig. 4. EditPerson frame.

Utilisation d'un "menu" :




```
menu: frame
   operation: (add_employee,edit_employee,add_
   supplier);
format
   center("Operation Selection Frame");
   space(2);
   input(operation, label: "Select operation:");
end;
```

```
for p in genf(menu, s) do
   switch op.operation

   case 'add_employee':
      append callf(newPerson, s) to PERSON;

   case 'edit_employee':
      x := callf(editPerson, s);
      update p in PERSON where p.name = x.name do
         replace p by x;
      end update;

   case 'add_supplier':
      append callf(new_supplier, s) to SUPPLIER;
   end switch;
end for;
```

2.7.4 - *Les grilles et le langage de programmation de la base de données*

Le déroulement du dialogue, c'est à dire l'enchaînement des grilles, se spécifie grâce aux structures de contrôle du langage Rigel associé à la base de données ; instruction CAS pour les grilles de type "menu", itérations sur la même grille ...

Tout programme de dialogue dispose également des primitives de gestion de la base de données.

Screen Rigel permet donc l'écriture d'applications de gestion d'une base de données utilisant des grilles d'écran. Les programmes ainsi créés sont indépendants d'un type de terminal particulier, mais surtout restent cohérents avec la base de données quelle que soit l'évolution de sa structure. La description des grilles à l'aide d'un langage de haut niveau et l'utilisation du dictionnaire de la base simplifie donc considérablement la programmation de telles applications.

2.8 - GESCRAN - CONSCRAN Actes des Journées BIGRE 82 - B. Meyer.

GESCRAN et CONSCRAN sont des logiciels développés au centre de calcul d'EDF à Clamart, sur un matériel IBM. Les programmes de ce centre sont en général écrits en FORTRAN. Le projet a pour but de permettre l'utilisation du contenu d'un écran entier comme unité de communication.

L'utilisation des "deux dimensions" de l'écran doit permettre :
- la visualisation du contexte de travail de l'utilisateur
- la correction éventuelle tant que la page n'est pas validée
- le remplissage automatique de certaines zones

Le travail a été facilité par les possibilités offertes par le sous-système conversationnel, qui permet :
- l'utilisation de l'écran pleine page
- la conservation d'une session à l'autre des caractéristiques de l'utilisateur
- un mode particulier pour le traitement des erreurs.

Ces différentes possibilités, utilisées par le système, sont mises à la disposition du programmeur.

2.8.1 - *Construction des grilles*

Deux progiciels facilitent la construction de grilles : GESCRAN, qui définit les grilles comme objets abstraits et CONSCRAN, qui permet le "dessin" des grilles directement sur console.

GESCRAN permet à un programme de définir et d'utiliser des grilles (écrans) par l'intermédiaire d'appels à des procédures FORTRAN. Les écrans sont définis en temps qu'objets abstraits : ils sont connus du programmeur par un nom et leur manipulation n'est possible que par les sous-programmes spécifiques de GESCRAN.

Un programme peut donc créer un écran, composé de fenêtres rectangulaires. Il définit ensuite leur contenu et caractéristiques : état initial, protection, brillance, couleur ... Il peut alors afficher cet écran et prendre en compte les données (typées) saisies par l'utilisateur.

La notion d'écran (grille logique) permet de dissocier les structures de données nécessaires de leur implémentation physique. Les écrans sont indépendants

d'un type de console donné (taille d'écran, opérations physiques d'entrées-sorties ...) Ceci doit donc rendre portables les programmes écrits à l'aide de GESCRAN.

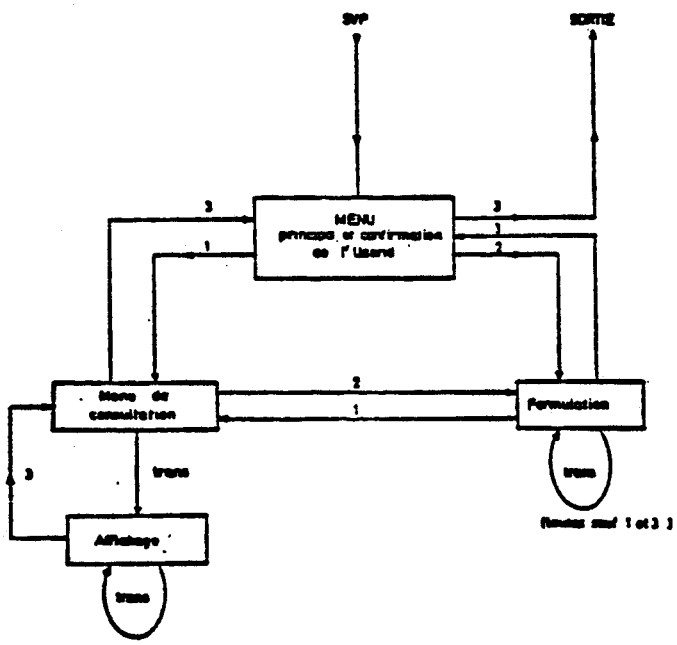
La définition d'un écran par une suite d'instructions étant une opération fastidieuse, un logiciel de définition interactive d'écrans a été développé. CONSCRAN permet le dessin des écrans sur console, leur archivage, les retours arrières pendant la construction ... CONSCRAN est une application écrite à l'aide GESCRAN et possédant donc toute la souplesse des dialogues "pleine page".

Les grilles utilisées par un programme étant définies, il faut ensuite régler leur enchaînement.

2.8.2 - *Structure des programmes de dialogue*

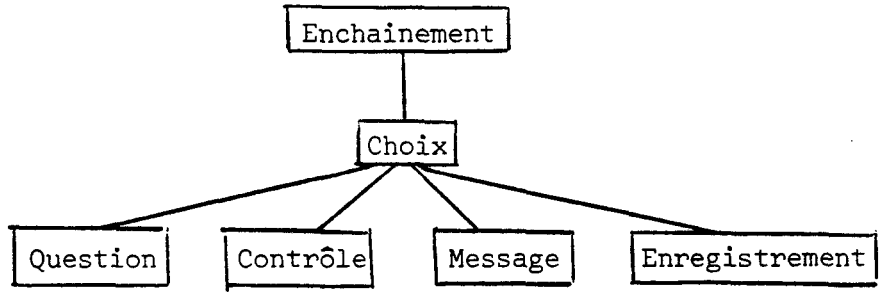
Les programmes de dialogue peuvent en général être modélisés par un graphe dont les noeuds sont des grilles et les transitions le passage d'une grille à une autre. Une exécution particulière du programme correspond donc à un certain cheminement dans ce graphe. Cette notion a été illustrée dans plusieurs articles parus dans "01 INFORMATIQUE" : Janvier 80, février et mai 81 ...
A chaque étape du dialogue, la grille associée est affichée, l'utilisateur saisit ses données et une série de contrôles est effectuée. Le passage à la grille suivante dépend des options prises par l'utilisateur : choix d'une touche de fonction ...

Exemple de graphe de dialogue



Les chemins possibles dans le graphe pouvant être très divers (retours arrière, touche aide, abandon ...), une programmation classique à l'aide d'instructions conditionnelles devient vite inextricable. Le graphe peut être programmé à l'aide d'une table : l'indiçage de la table par l'état courant et l'option choisie par l'utilisateur donne la transition, donc la grille à afficher. Le graphe, donc la structure du dialogue, n'est plus représentée par la structure du programme mais par une structure de données particulière : la table de décision.

Tout programme de dialogue pourra alors être défini par les données de ce tableau des transitions et les modules suivants :



Les modules enchainement et choix sont indépendants d'une application donnée. Le module choix utilise le tableau des transitions. Les quatre autres modules sont particuliers à un programme conversationnel donné : question affiche les grilles désirées, contrôle vérifie la "sémantique" des saisies, enregistrement exécute les actions désirées par l'utilisateur (mise à jour d'une base de données.

Pour construire cet outil, il faut disposer d'un langage permettant l'utilisation des types abstraits et la généricité : la description des objets manipulés par les modules doit pouvoir être suffisamment abstraite et les programmes doivent être paramétrés à un haut niveau. La notion d'état (noeud du graphe) est un exemple de structure abstraite. Le tableau de transition aura ce type "état" comme paramètre de généricité. Le langage SIMULA 67, avec sa notion de classe et de procédure virtuelle, a permis la mise en oeuvre de la méthode. On a donc une programmation très modulaire et descendante. Les parties indépendantes de l'application sont programmées séparément, la programmation d'un dialogue particulier consistant à une création d'exemplaires de ces modules précisant les caractéristiques propres à l'application.

Ce projet s'est donc particulièrement attaché à résoudre le problème de l'enchaînement des grilles. Une autre réalisation, LOGTRAN, a cherché à soulager le programmeur de la gestion des reprises.

2.9 - Le langage LOGTRAN B. Drieux - SERLIE

LOGTRAN est un langage basé sur un assembleur pour MICRAL.
Il fournit des directives (macro-instructions) facilitant :
- la structuration d'un programme en blocs de reprise
- la définition de grilles d'écran

Si les utilitaires de gestion de grilles d'écran sont assez nombreux , aucun langage évolué ne permet de gérer les reprises. La gestion du "conversationnel" est laissée à la charge du programmeur.

LOGTRAN, répondant à des problèmes précis d'applications qui "tournent" pourra être à la base d'une étude des reprises en cours de traitement (problème de restitution de contexte).

LOGTRAN implémente la notion devenue classique de grilles d'écran.

2.9.1 - *Les grilles d'écran en LOGTRAN*

LOGTRAN différencie les saisies purement conversationnelles des saisies par grille. Une saisie conversationnelle, caractère par caractère, est utilisée pour des données isolées à contrôler immédiatement (nom de mouvement, clé d'accés à un enregistrement ...). Dès que le nombre d'informations à saisir augmente, une saisie caractère par caractère devient fastidieuse : la grille permettra alors une saisie groupée. Un groupe est un ensemble de données dont l'ordre de saisie n'a pas d'importance. L'utilisateur peut revenir à tout moment sur un élément du groupe et modifier ce qu'il avait saisi. Chaque élément d'un groupe de saisie doit être accompagné d'un message indiquant à l'utilisateur ce que le programme attend de lui. L'ensemble des messages d'un groupe est affiché avant la saisie, en mode protégé (une modification de ces messages par l'utilisateur n'aurait pas de sens). Les saisies conversationnelles ayant servi d'introduction à une saisie groupée peuvent être présentes comme aide mémoire (visualisation du contexte). Elles devront alors être protégées. Les messages correspondant à ces saisies conversationnelles, ainsi que les messages de la saisie groupée pourront être affichés en une seule fois : c'est ce qu'on nommera le cadre de grille.

La partie statique d'une grille (le cadre) étant définie, il faut lui associer la définition de chaque élément de la grille :

- son type (conversationnel ou groupé)
- la variable associée, c'est à dire à laquelle la valeur saisie sera affectée
- le format de saisie (type, cadrage, limites éventuelles, saisie optionnelle ...)
- la zone de l'écran dans laquelle la saisie sera effectuée.

Cet ensemble (cadre et définition) constituera la grille de saisie.

2.9.2 - *La programmation d'une grille*

La programmation d'une grille n'est pas explicite dans un programme assembleur. Elle fait l'objet d'un texte source séparé constitué de directives et établi à l'aide d'un éditeur de texte.
Ce texte source est analysé et traduit par un utilitaire en un module de grille.
Le texte source définissant une grille comprend :
- la déclaration de la grille (nom ...)
- des messages "en-tête de cadre" éventuels
- des saisies conversationnelles éventuelles
- des saisies groupées éventuelles

Chaque groupe correspond à une sous-grille et doit avoir un nom, utilisé pour le remplissage ou la saisie.

Une partie éventuelle définissant une saisie en table : saisie en colonnes dont toutes les lignes ont la même structure.

Le nombre de lignes est soit constant, soit égal aux lignes restant sur l'écran.

Le programme utilisera les modules de grilles par l'intermédiaire d'appels de procédure. Plusieurs grilles pourront être affichées en même temps sur l'écran Le programmeur définit une hiérarchie sur les grilles grâce à un numéro de niveau dans leur déclaration. Le niveau d'une grille sert à déterminer la "présence ou l'absence" de son cadre sur l'écran à un moment donné de l'exécution du programme au moment de l'affichage du cadre d'une grille de niveau n, tous les cadres des grilles de niveau supérieur à n sont effacés. Plus une grille est "profonde" (bas niveau hiérarchique donc numéro de niveau élevé), plus son cadre s'affichera vers le bas de l'écran. La hiérarchie des grilles peut se représenter par une structuration en arbre. Dans un langage à structure de blocs de type ALGOL, le niveau d'une grille pourrait être le niveau du bloc contenant sa déclaration.

La notion de grille dans LOGTRAN est donc assez classique. LOGTRAN introduit néanmoins deux idées nouvelles : la hiérarchisation des grilles en arbre ainsi que la différenciation des saisies conversationnelles et groupées. L'intérêt principal de LOGTRAN réside dans l'implémentation des mécanismes de reprise.

2.9.3 - *Le concept de reprise dans LOGTRAN*

Lorsque le programmeur veut permettre une reprise du traitement, il crée un bloc de reprise : le début de ce bloc est le point de reprise privilégié pour toute reprise déclenchée par l'utilisateur ou le matériel à l'intérieur du bloc.

LOGTRAN prend comme principe que le traitement de reprise sur un bloc est le même que le traitement en cas de fin normale de ce bloc ; ce traitement consiste à restituer le contexte initial du bloc. Ceci vient de la définition d'un bloc dans LOGTRAN : chacune de ses ré-exécutions globales doit s'effectuer dans le même contexte initial que les précédentes. Par exemple, un bloc ayant réservé des enregistrements doit les libérer, un bloc ayant ouvert un fichier doit le refermer ...

Cette définition d'une reprise (restitution du contexte initial identique à une fin normale) limite le contexte au seul contexte "externe" : état des paramètres de gestion du fichier ou de la base de données manipulé. Le contexte "interne" (valeur des variables, valeur des enregistrements ...) en est exclu.

En effet, vis à vis du contexte interne, une reprise devra restituer le contexte initial, c'est à dire retrouver les valeurs avant tout traitement du bloc, alors qu'une fin normale du bloc devra évidemment laisser inchangées les nouvelles valeurs acquises dans le bloc.

Quels outils LOGTRAN propose-t-il au programmeur pour définir ses blocs de reprises ?

2.9.4 - *Les blocs de reprise*

Le langage LOGTRAN ne possède pas de structure de blocs au sens ALGOL. Deux autres formes de structuration apparaissent :

- structuration en segments de recouvrement
- structuration en blocs de reprise.

Chacun des différents types de bloc de reprise répond à un problème précis. On distingue :

- les blocs ordinaires, ne comportant pas de saisie
- les blocs primaires (transformations effectives en fin de bloc) et les blocs secondaires (tranformations propagées au bloc englobant)
- les blocs conditionnels, permettant un traitement spécifique sur entrée vide
- les pseudo-ouvertures de bloc, permettant de différencier les reprises sur entrée vide des reprises sur données invalides
- les blocs conversationnels, liés aux saisies conversationnelles.

2.9.4.1 - Les blocs ordinaires

Un bloc ordinaire ne contient pas de saisie. L'ouverture d'un bloc ordinaire se justifie par la possibilité de pannes matérielles ou de tentatives d'accès à un enregistrement déjà réservé. Le traitement de reprise pourra en avertir l'utilisateur : choix entre mise en file d'attente ou abandon de la transaction ...

Dans un langage à structure de bloc, un bloc ordinaire pourrait être un bloc contenant un traitement d'exception de type ADA.

2.9.4.2 - Les blocs non ordinaires

Ce sont donc des blocs contenant des saisies. On distingue les blocs primaires des blocs secondaires. En cas de fin normale d'un bloc primaire, les traitements de reprise (liés à la gestion de fichier ...) sont exécutés. En cas de fin normale d'un bloc secondaire, ces traitements sont propagés au bloc immédiatement englobant. Cela permet, par exemple, de ne pas libérer un enregistrement qui sera à nouveau utilisé par le bloc englobant.

Dans un langage à structure de bloc, le choix entre libération effective ou propagation au bloc englobant pourrait se faire en fonction du lieu de déclaration de l'objet externe considéré [CARRE 80].

2.9.4.3 - Les blocs conversationnels

LOGTRAN différencie les saisies groupées des saisies conversationnelles (caractère par caractère). Les saisies conversationnelles concernent des données de définition (nom de mouvement, clés d'accès ...), les saisies groupées des données de traitement (mise à jour d'un enregistrement ...). Ces deux types de données sont de nature différente. Les données de définition d'un traitement sont saisies et contrôlées dans un ordre précis. Leur chronologie correspond à une

certaine hiérarchie (du général vers le particulier) et permet de cerner de plus en plus précisément l'ensemble des données mise en jeu par le traitement. Il n'est pas logique (sauf cas particuliers) de définir l'enregistrement concerné par une transaction avant de définir le type de la transaction. En cas de clé d'accés multiple, le contrôle de validité impose un ordre de saisie (par exemple : ville, rue, bâtiment, étage, appartement). Enfin, les données de définition doivent être contrôlées avant la saisie des données du traitement. La saisie dans l'ordre logique peut éviter des saisies inutiles : inutile de donner les nouvelles valeurs d'un enregistrement si l'accés en mise à jour sera refusé.

Bien que ces deux types de saisie ne soient pas du même niveau hiérarchique, on pourrait admettre qu'une saisie conversationnelle se face par une sous-grille réduite à un élément, le programme de dialogue assurant les contrôles nécessaires.

2.9.4.4 - Les pseudo-ouvertures de bloc

permettent de mettre sur un même plan (même niveau d'emboitement de bloc) plusieurs saisies conversationnelles. On peut ainsi avoir un lieu de reprise différent suivant qu'une donnée d'accès n'est pas valide ou que l'utilisateur déclenche la reprise par une entrée vide (reprise au début de la chaîne d'acc s).

2.9.4.5 - Les blocs conditionnels

contiennent une saisie pour laquelle l'entrée vide n'est pas autorisée mais est significative. L'entrée vide entraîne alors l'exécution d'un traitement de reprise spécifique. C'est le cas pour les tableaux flexibles (nombre de lignes non connu à l'avance) : entrée vide pour indiquer la fin de liste.

LOGTRAN, langage d'écriture du dialogue utilisant des grilles d'écran, s'est donc attaché à résoudre les problèmes de reprise rencontrés lors de la programmation d'applications manipulant des données externes (fichiers) partagées entre plusieurs utilisateurs. Les différents types de bloc de reprise pourraient être l'implémentation du mécanisme de reprise proposé par un langage à structure de bloc.

Suite à l'implémentation de LOGTRAN, une réflexion plus approfondie sur la programmation d'applications conversationnelles a été menée par CARREZ [CARRE 80].

2.10 - Le Concept de reprise d'après [CARREZ 80]

L'extension des possibilités de traitement en temps réel pose le problème de l'écriture de programmes conversationnels. L'utilisateur peut à tout moment influencer le déroulement du programme, et les langages de programmation ne sont pas adaptés à l'écriture des différentes possibilités qui lui sont offertes. Par exemple, la présence "en-ligne" de l'utilisateur doit lui permettre de corriger les erreurs, qu'il faudra donc détecter en temps réel.

2.10.1 - *Les données échangées et les cas de reprise*

Les données fournies par l'utilisateur sont de trois types :
- les données de définition, précisant la transaction demandée
- les données d'accès, définissant les objets concernés par cette transaction
- les données du traitement, définissant les modifications à apporter à ces objets.

Lors d'une demande de donnée par le programme, le contrôle de l'exécution par l'utilisateur se traduit par quatre possibilités :
- entrer la donnée
- revenir sur des données déjà saisies
- abandonner la transaction en cours
- signaler l'épuisement de données en cas de listes.

Les deuxième et troisième possibilités correspondent au cas de reprise, l'abandon de la transaction étant une reprise sur le code de la transaction. Pour le programme, il ne suffit pas d'effectuer une nouvelle affectation, mais il faut déterminer le point de reprise et remettre son contexte dans un état adéquat ; annulation des réservations de données partagées et des modifications des variables qui ont eu lieu après le point de reprise ...

Ce point peut être déterminé implicitement ou à l'aide d'un dialogue. L'association implicite d'un point de reprise à chaque saisie allège l'effort de programmation. Pour un langage à structure de blocs, ce point pourra être le début du bloc englobant la saisie, ce qui permet la gestion du contexte en pile.

Une reprise peut également être provoquée par le programme, lorsqu'il détecte une erreur. Ces erreurs sont de plusieurs types :

- erreur mineure ne nécessitant pas de reprise
- erreur non liée au contexte interactif du programme : mise en file d'attente ou abandon lorsqu'une donnée partagée n'est pas disponible ...
- erreur nécessitant la resaisie d'une donnée
- erreur majeure entrainant l'abandon de la transaction : incohérence de fichiers.

Le même mécanisme de gestion de contexte en pile pourra servir aux reprises en cas d'erreur.

2.10.2 - *Structure de bloc et restitution de contexte*

La notion de reprise étant associée à la notion de bloc, on distingue la fin normale du bloc (traitement interne devenant effectif) de la fin anormale (annulation des traitements internes par restitution du contexte initial) :

Cette restitution présente trois aspects différents :
- objets internes au programme
- objets externes
- synchronisation avec d'autres processus.

Exemple :

```
a) début saisie typetransaction
b)     | cas type transaction de
c) création : répéter début saisie codemodèle
d)          |          | si codemodèle existe alors message 'existe déjà'
e)          |          | sinon saisie, vérification des paramètres
f)          |          |       mémorisation du modèle              fsi
g)          |          | cptmodèle : = cptmodèle + 1
h)          |          | répéter jusqu'à fin de liste début saisie codematériau 1
i)          |          |     | si codematériau 1 n'existe pas alors
j)          |          |     |          erreur 'n'existe pas' fsi
k)          |          |     | saisie des autres codes matériaux
l)          |          |     | cptprod : = cptprod + 1
m)    B1    |          |     | début allouer espace fichier
n)          |    B2    |  B3 |        impossible : erreur 'pas d'espace':
            |          |     |   B4
p)          |          |     |    | saisie des paramètres et mémorisation
q)          |          |     | fin
r)          |          | fin
s)          |    fin   |
t)          |          | autres cas
u)    fin
```

2.10.2.1 - Objets_internes

On peut créer une copie en tête de bloc, qui utilisera alors cette copie. Si le bloc est abandonné, on détruit la copie, et s'il se termine normalement la version antérieure est remplacée par la copie. A chaque ouverture de bloc, il y a création de copies, gérées en pile. La reprise en début de bloc consiste à détruire la copie associée et les copies de niveau supérieur dans la pile. Pour économiser la place mémoire, la copie peut être limitée aux objets modifiés dans le bloc ou les blocs emboîtés . En début de bloc, on crée des objets locaux au bloc pour chacun des objets à recopier, et on leur assigne les valeurs qu'ils ont à l'extérieur. En fin normale du bloc, on assigne les valeurs de ces objets locaux aux objets correspondants.

2.10.2.2 - Objets_externes

Ces objets, partagés entre plusieurs utilisateurs, imposent une synchronisation des accès. Ils possèdent des propriétés de cohérence que chaque transaction doit respecter. Le déroulement d'une transaction pouvant entrainer une incohérence momentanée, celle-ci vérouillera les données pendant la durée de l'infraction aux règles de cohérence. L'abandon d'une transaction ne doit pas entrainer d'incohérence, ni bloquer les enregistrements verrouillés. Comme pour les données internes, il suffira d'effectuer les mises à jour sur des copies attachées aux blocs. En cas d'abandon, ces copies seront détruites et les enregistrements libérés. En cas de fin normale, deux solutions sont à envisager :

- si les objets externes sont déclarés dans le bloc, les modifications deviennent permanentes
- si ces objets ont été déclarés dans des blocs englobants, les modifications sont intégrées à celles du bloc immédiatement englobant.

Ce mécanisme respecte la cohérence des données à la condition que des données liées par une règle de cohérence soient déclarées dans le même bloc.

2.10.2.3 - Partage des objets_après_leur modification

Le maintien de la cohérence implique qu'un objet externe ne soit libéré que lorsqu'il ne sera plus modifié par la transaction. Mais on ne peut pas appliquer cette règle aux informations de gestion des fichiers : une transaction de type "création d'enregistrements" monopoliserait alors ces fichiers.

Ces informations de gestion de ressources ne peuvent donc pas être gérées par copies : en cas de reprise, il ne s'agit pas de leur redonner leurs valeurs antérieures (d'autres transactions ont pu les modifier depuis) mais il faut effectuer un traitement spécifique, qui assurera, par exemple la restitution à l'espace libre des enregistrements créés. Ce traitement de reprise, suffisamment systématique, peut faire l'objet d'une procédure externe au programme.

La gestion du contexte en pile permet donc de décharger le programmeur de la définition explicite des reprises. Les paramètres de gestion de ressources externes devront quand même être traités d'une manière spécifique.

2.11 - Conclusion

La notion de grilles d'écran est donc acceptée par tous les concepteurs de logiciel d'aide à l'écriture de dialogues alphanumériques. En prenant comme unité de communication non plus le caractère ni même la ligne, mais l'écran ou une partie d'écran, elle assouplit l'écriture des programmes et leur utilisation.

De nombreux avantages ont été mis en valeur :
- les utilisateurs sont habitués aux formulaires-papier, et ne sont donc pas déroutés par le mode de saisie en grille (vitesse et fiabilité des saisies)
- le contexte de travail de l'utilisateur est visualisé
- la correction reste possible tant que la grille n'est pas validée
- une grille permet de regrouper les entités logiquement dépendantes (modularité)
- la notion de grille permet de disssocier les messages des structures de données
- le groupement facilite l'écriture, la re-lecture et la modification des programmes (réduction du coût logiciel)
- les applications, ne tenant plus compte d'un matériel particulier, deviennent portables
- l'implémentation sur un matériel particulier peut être très efficace, sous réserve de fournir un "traducteur" performant.

Les critères de comparaison entre les différents logiciels proposés peuvent se classer en deux grands groupes :
- facilité d'écriture et de maintenance des programmes
- souplesse du dialogue utilisateur.

Une étude comparative est donnée par le tableau suivant. Ce tableau comparatif est à prendre sous toutes réserves :

- il réunit des logiciels difficilement comparables : langages (Pascal interactif), systèmes (GESCRAN - CONSCRAN), utilitaires moins évolués (LOGTRAN). Il est par exemple évident que la sauvegarde du contexte lors de l'abandon temporaire d'une session est du domaine "système"

- ce tableau ne mentionne pas certaines caractéristiques particulières à chacun des logiciels ; Screen Rigel utilise, par exemple, le dictionnaire de la base de données pour permettre une mise à jour de la structure de la base sans ré-écriture des grilles

- ce tableau est incomplet, par manque de documentation : peut-on utiliser des grilles en table avec GESCRAN - CONSCRAN ?

- enfin, certains critères étant subjectifs, ce tableau est arbitraire : à partir de quand une grille ne doit-elle plus être considérée comme statique ?

LOGICIELS ou ETUDES	Facilité de définition des grilles							Aspect Utilisateur				Déroulement du dialogue			Contrôles		Déroutements conversationnels		Adaptation à l'utilisateur			Portabilité		
CRITERES	comptage automatique des lignes et colonnes	sous grilles	grilles en table	options par défaut	Editeur d'écran interactif	Séparations structure données/messages	Définition de contraintes sémantiques	justification automatique	contrôle syntaxique à la volée	fenêtre d'écran	remplissage automatique de champs	Grilles Statiques ou Dynamiques	Séparation dialogue/traitement	Enchaînement des grilles	contrôle syntaxique	contrôle sémantique	aides	reprises	conservation d'un profil utilisateur	niveaux (débutant, confirmé ...)	sauvegarde du contexte pour abandon temporaire	indépendance taille d'écran	intégration à une base de données	indépendance matériel
DFC					oui (-)	oui (-)	par pgr	oui	oui		oui	S		App Proc	fbe	pgr								oui (-)
Logiciel IUT Lille	oui			oui	oui	oui (-)		oui	oui		oui	S		App Proc	fbe	pgr								
Interrogip II			oui				pgr	oui	oui		oui	S		App	fbe	pgr		fbe					oui	
QBE - SBA	oui	oui	oui						oui		oui	S		par ent									oui	oui
Pascal Interactif	rel	oui	oui			oui	oui		oui		oui	D		App	oui	oui		fbe						oui
Screen Rigel	rel		oui	oui		oui	pgr		oui			S		App	oui	pgr							oui	
GESCRAN-CONSCRAN	oui				oui	oui			oui	oui	oui	D		gra	oui	dif ?	oui	fbe	oui			oui		oui
LOGTRAN		oui	oui			oui	pgr		oui			S		App	oui	pgr		oui					fich	
CARREZ		oui				oui	oui		oui			D		imp	oui	oui		oui				oui		oui
GEHANI	oui					oui	oui		oui		oui	S		app	oui	oui		oui				oui		oui

App : Appels - fbe : faible - par ent : paramètres d'entrées - rel : relatif - gra : graphe - dif : difficile - fich : fichiers - imp : implicite -

COMPARAISON DES LOGICIELS D'AIDE A L'ECRITURE DE DIALOGUES

Malgré toutes les réserves annoncées, ce tableau présente l'intérêt de dégager les grandes tendances des applications actuelles. Un "tronc commun" semble acquis : séparation structures de données/messages, contrôle syntaxique ... Par contre, les contrôles sémantiques et les facilités de programmation des reprises sont assez balbutiants.

Une grille est le plus souvent statique : la grille est affichée en bloc au moment de son appel et disparait après les saisies nécessaires. La possibilité d'incorporer des grilles définies par ailleurs (sous-grille) évite des répétition fastidieuses.

Pascal interactif propose des grilles plus dynamiques : une sous grille peut s'afficher en cours de traitement de la grille dont elle fait partie, et, par le jeu des règles de comportement, l'apparence d'une grille peut totalement se modifier. Mais cette "dynamique" provient d'actions et de règles d'interaction explicitement prévues par le programmeur et non pas de la structure générale du programme.

Certains logiciels incluent la description du cadre de grille dans le programme d'application. Mais l'intérêt d'y trouver les messages explicitant le rôle des variables en saisie ne compense pas la perte de lisibilité de la structure du programme. Encore que, pour Pascal interactif, par exemple, les possibilités de modularité pourront éviter cet écueil : association grille/ procédure ...

Le programmeur n'a plus à tenir compte de la mise en écran au niveau du caractère (nombre d'espaces ...) mais doit encore gérer les lignes et colonnes. Seul CONSCRAN propose un éditeur de masques d'écran interactif : la description de la grille d'écran ne se fait plus par un ensemble d'instructions mais sur l'écran lui-même. Quant à Gehani [GEHAN 83], il propose l'utilisation d'une heuristique qu'il n'explicite pas. Cette possibilité est pourtant essentielle : elle permet au programmeur de ne s'attacher qu'à l'aspect "structure des données" d'une grille au moment de l'écriture du programme et facilite la modification ultérieure des messages.

Mais pour tous les logiciels, les grilles sont traitées comme des modules indivisibles : on appelle une grille, on attend que l'utilisateur ait terminé la saisie des éléments, puis on retourne au "programme appelant"

ou on effectue la transition vers une autre grille avec exécution des traitements demandés par l'utilisateur. A ce moment, la grille est validée, disparait de l'écran et aucun retour n'y est plus permis, sauf actions contraires prévues par le programmeur. Les logiciels diffèrent donc seulement par la possibilité d'intégrer ou non des traitements aux grilles : contrôles "sémantiques" ...

Ce schéma à trois phases (affichage de la grille/saisie-vérification/traitement-effacement), relativement simple (voire simpliste ?) suffit pour la plupart des transactions de gestion (création d'un enregistrement ...) Pour les transactions un peu plus complexes, un ensemble restreint de grilles peut être nécessaire. Mais ce schéma n'est pas du tout adapté aux programmes non transactionnels nécessitant des saisies ou des éditions.

La structure du dialogue, c'est à dire l'enchaînement entre les saisies et éditions, est en effet beaucoup plus difficile à maitriser que la définition d'une grille-transaction élémentaire.

GESCRAN-CONSCRAN apporte une solution originale à ce problème : le problème est un graphe de transition entre les grilles. Une table de décision (représentant le graphe) est certainement beaucoup plus lisible qu'une cascade de structures de contrôle. Mais tous les traitements doivent alors être incorporés aux grilles.

Suivant les logiciels, une plus ou moins grande confusion est donc faite entre les différents niveaux de définition d'une application interactive :
- structure des données en saisie et en édition
- messages associés
- déroulement du dialogue, c'est à dire affichage/effacement des cadres de saisie
- traitements associés.

La conclusion de cette bibliographie peut être que les constructeurs d'aide à l'écriture de dialogue restent prisonniers du schéma de saisie sur formulaire, hérité de la pratique classique du travail de bureau. Grâce aux moyens informatiques, ce schéma a pu être rendu beaucoup plus séduisant ; les formulaires sont maintenant "intelligents". Vis à vis du système, l'application passe d'un état "saisie de formulaire" à un autre, les actions étant, suivant les logiciels, incluses dans le formulaire ou dans les transitions. En fin de formulaire, il y a validation sans retour possible (du moins simplement).

Les grilles sont donc des entités disjointes, ce qui pose le problème, évoqué par Gehani [GEHAN 83] des liaisons entre les champs des grilles différentes. Les grilles sont plus considérées comme structures de données seules que comme des structures de données associées à leur traitement. Ceci est moins vrai dans l'approche "types abstraits de données", où une grille est un ensemble de valeurs plus les opérations permises sur ces valeurs. Mais peut-on encore parler de type abstrait lorsque les opérations font intervenir des données externes à la grille (autres grilles ou base de données ...)

Ce schéma "grille-formulaire" est tout à fait approprié lorsque c'est l'utilisateur qui définit la transaction (SBA, logiciel IUT). Mais les contrôles et opérations possibles sont alors limités.

Lorsque l'application est programmée par un informaticien, des outils moins immédiats mais beaucoup plus évolués peuvent être proposés.

En particulier, aucun logiciel n'a proposé un mécanisme global de traitement des erreurs de saisie. Pourtant, la principale difficulté de la programmation de l'interaction homme/machine vient de ces traitements d'erreur. A tout moment, le programme peut être dérouté de son exécution "normale", soit suite à une panne matérielle, mais bien plus souvent par demande de reprise de l'utilisateur ou du programme sur des données déjà saisies.

Le langage qui sera proposé dans les chapitres suivants propose l'utilisation de la structuration en blocs d'un programme d'application pour définir et contrôler le déroutement du dialogue nécessaire à l'exécution de ce programme.

Le programmeur s'attachera donc à structurer convenablement son programme (passage du général au particulier par blocs imbriqués ...). Cette structuration logique définira :
- l'affichage et la disparition des structures de saisie (grilles dynamiques)
- les possibilités de reprise.

La description des reprises sera donc incluse dans la structure du programme et ne nécessitera pas d'instructions ou de structures de contrôle explicites.

La notion de "formulaire-état" disparait donc, laissant place à une vision plus continue et dynamique d'une application. En particulier, une

"grille" ne sera plus une entité figée mais l'ensemble des structures de saisie affichées à un moment donné. On parlera donc plus de contexte courant que de grille.

3 - CONCEPTION D'UN LANGAGE DE DIALOGUE

Un bon langage de dialogue devra atteindre deux objectifs parfois contradictoires :
- ce langage devra permettre l'écriture de dialogues répondant aux désirs de l'utilisateur. Les caractéristiques attendues par celui-ci d'un "bon" interface conversationnel ont été explicitées dans la première partie.
- cet outil devra limiter au maximum le travail du programmeur.
En effet, le programmeur dispose déjà de moyens suffisant pour écrire de bons programmes conversationnels : consoles avec mode "pleine page", langages de programmation permettant le traitement des erreurs par branchements spécifiques .. En utilisant la notion de procédure, le programmeur peut également séparer dialogue et traitement ; il peut donc écrire des programmes conversationnels relativement souples à l'utilisation et évolutifs. Pourquoi fournir au programmeur un langage spécifique d'écriture de dialogue ?

3.1 - Un langage de dialogue ?

Les outils existants ne sont pas adaptés aux applications interactives. Par exemple, les cascades de ruptures de séquence pour traiter les différentes options et cas d'erreurs aboutissent à des programmes illisibles, à la mise au point problématique et interdisant toute modification ultérieure.

Un exemple de procédure de dialogue, écrit en ADA, montre le coût de l'écriture de possibilités d'interaction. Encore faut-il souligner que le langage ADA introduit des concepts puissants et des mécanismes de haut niveau tels que les exceptions.

Dans l'exemple suivant [MR 80], l'utilisateur sélectionne une couleur, et le programme lui affiche le nombre correspondant disponible en stock.

```
procedure DIALOGUE is
  use TEXT_IO;
  type COLOR is (WHITE, RED, ORANGE, YELLOW, GREEN, BLUE, BROWN);
  INVENTORY : array (COLOR) of INTEGER := (20, 17, 43, 10, 28, 173, 87);
  CHOICE : COLOR;
  package COLOR_IO is new ENUMERATION_IO(COLOR); use COLOR_IO;

  function ENTER_COLOR return COLOR is
    SELECTION : COLOR;
  begin
    loop
      begin
        PUT("Color selected: ");
        GET(SELECTION);
        return SELECTION;
      exception
        when DATA_ERROR =>
          PUT("Invalid color, try again. ");
      end;
    end loop;
  end;
begin  --  body of DIALOGUE;
  CHOICE := ENTER_COLOR();
  NEW_LINE;
  PUT(CHOICE, LOWER_CASE => TRUE);
  PUT(" items available: ");
  SET_COL(25);
  PUT(INVENTORY(CHOICE), WIDTH => 5);
  PUT(";");
  NEW_LINE;
end DIALOGUE;
```

Example of an interaction (characters typed by the user are italicized):

```
Color selected: black
Invalid color, try again. Color selected: blue
blue items available:      173;
```

Dans cette procédure, le programmeur a défini :

- les mises en page (ou en écran) : messages, passages à la ligne ...
- les erreurs et messages d'erreurs
- la recherche d'une valeur dans une structure de données qui sera externe au programme en situation réelle (l'inventaire).

Cette procédure reste lisible, mais les choix sont peu nombreux. Que donnerait un exemple plus réaliste ?

Un langage de dialogue devra donc proposer des mécanismes de haut niveau, particulièrement pour résoudre le problème de reprise en cas d'erreurs.

Le travail de construction d'un dialogue peut se décomposer en différentes couches :

- définition de la structure générale du dialogue (enchaînement et traitements indispensables)
- définition de l'affichage : description, si possible interactive, du contenu de chaque écran ou portion d'écran
- traitements externes au dialogue.

L'importance relative de ces différents niveaux et leur ordre d'élaboration peut dépendre du type de programme, de la méthode d'analyse et de ce qui existe déjà.

3.2 - Relation entre dialogue et traitement suivant le type de programme

Quel que soit le type de programme, les procédures conversationnelles serviront trois propos :

- déterminer le type de traitement demandé par l'utilisateur
- déterminer les données de ce traitement
- présenter, le cas échéant, les résultats du traitement

On peut considérer tout programme comportant des saisies-édition comme deux processus coopérant à la même tâche : côté machine, le processus "traitement", côté utilisateur, un processus "dialogue". Cette séparation peut amener un certain parallélisme : saisie par anticipation ... Il faut alors disposer d'outils de synchronisation et de communication adéquats.

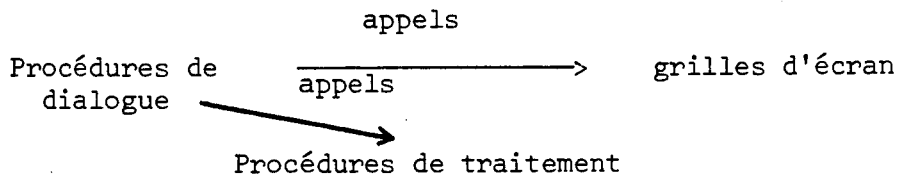
On peut sans doute trouver des applications où ces deux "co-routines" ont la même importance. Mais en général, l'une ou l'autre sera privilégiée : leur fonctionnement sera hiérarchisé.

Dans un programme de type scientifique, le dialogue ne sera que l'introduction et la conclusion au programme proprement dit. Par exemple, après avoir demandé en bibliothèque le module d'inversion de matrice par une méthode adaptée à son problème, l'utilisateur entrera sa matrice sur console. Le travail principal, c'est à dire l'inversion de matrice, pourra alors s'exécuter. Le traitement, de niveau hiérarchique plus élevé, utilise donc des procédures annexes de saisie de données et d'édition des résultats.

Un tel programme peut se schématiser par les trois niveaux :

Programme de traitement —appels→ Procédures de Dialogue —appels→ Grilles d'écran

Pour un programme de gestion, la partie dialogue devient prépondérante. L'exemple type sera la manipulation d'une base de données. L'utilisateur demandera une suite de transactions (consultation, mise à jour ...). Le niveau traitement pourra se résumer à des appels aux primitives du système de gestion de base de données. Ceci peut également se schématiser en 3 niveaux :



Ce sont donc les procédures de saisie-édition qui détermineront le déroulement du programme.

L'analyse d'un problème de gestion peut d'ailleurs consister à écrire les différentes grilles associées aux transactions.

Les programmes de gestion, comme les programmes scientifiques, peuvent donc profiter des mêmes outils de haut niveau d'aide à l'écriture de dialogue. Leur différence fondamentale réside dans l'importance relative du dialogue et du traitement. Dans le domaine scientifique, c'est le traitement qui impose l'ordre d'exécution des routines d'entrées-sorties, alors que pour les problèmes de gestion, les routines de traitement ne sont que des primitives à la disposition du programme de dialogue.

Le degré de dépendance d'un niveau sur l'autre pourra déterminer les possibilités de reprise laissées à l'utilisateur. Un programme conversationnel devra envisager des reprises d'exécution provoquées par celui-ci:une donnée déjà saisie et acceptée par le programme était erronée, ou plutôt était valide mais ne correspondait pas à ce que l'utilisateur voulait faire. Les programmes de type scientifique correspondent à une connexion faible entre les procédures de dialogue. On peut envisager de limiter les reprises à l'intérieur d'une procédure. Les programmes de type gestion doivent sans doute être plus souples : les procédures de dialogue pouvant s'appeler entre elles, une reprise provoquée sur une procédure terminée peut s'avérer utile. Mais les problèmes liés aux

données partagées obligent à créer des points de validation : toute donnée saisie ne peut être reprise après l'exécution d'une validation.

Les modalités de reprise devront donc laisser au programmeur le choix entre différents mécanismes. La structuration en blocs des procédures de dialogue va faciliter la mise en oeuvre de ces mécanismes. Un éditeur de grilles d'écran interactif facilitera la "mise en écran" des données échangées.

3.3 - Editeur de grilles d'écran

Du point de vue du dialogue, il existe trois familles de données, correspondant au sens de l'échange homme/machine :

- les données en saisie
- les données en édition
- les données en édition pour mise à jour (édition-saisie).

Chacune des données échangées a un nom et un type. Le nom a en fait deux aspects : le nom externe, connu du programmeur, auquel il faudra associer un nom interne pour la manipulation par le programme. Le type impose une syntaxe sur la donnée : entier, chaîne de dix caractères, "réel" compris entre zéro et cent ... Le nom et le type de chaque variable échangée étant définis, soit parce que le programme de dialogue est déjà écrit, soit parce que l'analyse du problème a déterminé la structure des grilles à afficher, il faudra lui associer un certain nombre d'attributs d'affichage. Un éditeur d'écran sera un outil interactif de mise en écran des variables échangées.

L'éditeur va permettre au programmeur d'associer à chaque variable une rubrique ou structure de saisie-édition, et gérer une bibliothèque de ces rubriques.

Le mécanisme de définition d'une rubrique pourra être le suivant :

- l'éditeur affiche le nom de la variable, son type et le sens de l'échange
- le programmeur lui associe alors
  - . un texte : message précisant à l'utilisateur la "sémantique" de la variable : question pour les variables en saisie ...
  - . une position, ou plus exactement le contour nécessaire à la clarté de l'affichage. L'emplacement de cette "fenêtre" dépendra du contexte d'affichage et n'est donc pas à définir.

. des caractéristiques d'édition et de formatage : insertion automatique de caractères, remplissage de blancs, de zéros, cadrages ...

La date 13/04/1983 pourra par exemple être rentrée comme 13/4, le "/" s'affichant automatiquement.

. un message d'erreur de syntaxe, si l'éditeur n'a pas encore de message associé au type de la variable.

Pour les variables de type ensemble, l'option par défaut est l'affichage des différents éléments avec choix par positionnement du curseur. Le programmeur pourra modifier l'ordre d'affichage et le nom des éléments.

L'ensemble des parties fixes d'une rubrique forme le cadre de saisie.

A l'exécution, il sera affiché au moment de la saisie (ou de l'édition), à la suite des cadres déjà affichés, avec disparition des cadres du haut d'écran en cas de débordement.

Une rubrique est associée à une variable individuelle. Certaines variables peuvent être groupées dans la même instruction de saisie ou d'édition. L'ensemble des cadres individuels formera un cadre de sous-grille, auquel on pourra associer un titre et un nom.

On peut distinguer trois types de sous-grilles :

- les sous-grilles de longueur fixe. Le nombre de variables qui la composent est constant. Ces variables pourront être de types différents.
- les sous-grilles dynamiques : le nombre de variables dépend d'une valeur connue au moment de l'affichage du cadre de sous-grille
- les sous-grilles flexibles : la sortie de la sous-grille est l'effet d'une saisie particulière (par exemple l'entrée vide).

Ces deux dernières seront composées de rubriques de même type (sous grilles multiples ou itératives).

Les sous grilles étant séparées par des traitements, elles ne pourront être affichées simultanément. S'il le désire, le programmeur pourra toujours retarder certains traitements (par exemple des vérifications de cohérence) pour permettre l'affichage groupé. Certaines contraintes seront imposées aux fenêtres associées aux rubriques et sous-grilles.

Pour les rubriques simples (une seule donnée, un enregistrement à quelques champs ...), il semble évident que la fenêtre soit suffisamment large pour permettre l'affichage de toute la rubrique. Le programmeur pourra au besoin l'élargir pour mettre en évidence cette rubrique.

Le problème est différent en ce qui concerne les sous-grilles. Si la sous-grille n'est pas répétitive, la fenêtre associée sera, a priori, la réunion des fenêtres associées aux rubriques internes. Les sous-grilles répétitives pourront être statiques, dynamiques ou flexibles. La fenêtre associée à un ensemble de rubriques de même type devra contenir, au minimum, une rubrique. Dans tous les cas, elle en contiendra un nombre entier. On conviendra que la sous-grille défilera "en rouleau" à l'intérieur de cette fenêtre. Le programmeur pourra également choisir l'option par défaut sans définir de fenêtres. La sous-grille multiple s'affichera alors "in extenso".
Ce mécanisme soulage entièrement le programmeur de tout comptage de lignes ou autres tâches fastidieuses. De plus à l'exécution, l'encombrement sur l'écran d'une telle sous-grille pourra être limité. Le contexte (les autres sous-grilles) pourra donc rester affiché.

La possibilité de déclarer des sous-grilles multiples n'est qu'un peu de "sucre syntaxique" pour condenser l'écriture des programmes. On pourrait se contenter de simuler une grille multiple à l'aide d'une itération explicite :

saisie ([100n] de rubrique) par Pour i = 1 à n début saisie (rubrique) fin

Il apparait alors judicieux de permettre l'association de fenêtres aux blocs de programme. Cette association d'attributs d'affichage (messages et fenêtres) aux blocs ou procédures sera explicitée en fin de chapitre. Pour permettre cela, l'éditeur d'écran devra offrir au programmeur la possibilité de lister le programme et de désigner un bloc particulier.

La disposition du texte de l'application au moment de la mise en écran offre une autre possibilité intéressante : le programmeur peut alors associer aux rubriques des messages comportant des variables du programme.
Exemple de message dynamique :

"Paramètres du "K"ième modèle ?"

Ceci évite d'écrire dans l'application elle-même des instructions d'édition qui n'auraient qu'un but informatif. On dissocie alors les éditions liées au traitement des éditions liées aux messages.

Les notions de rubrique de saisie et de sous-grille étant définies on peut se demander ce que recouvre la notion de grille. Une grille sera

l'ensemble des cadres affichés à un moment donné. Elle sera liée à une exécution particulière d'un programme particulier, donc entièrement dynamique ; la notion de grille est fortement liée à l'écriture des procédures de dialogue.

3.4 - Les procédures de dialogue

Après avoir défini ce qu'est une procédure de dialogue, il faudra donner les caractéristiques d'un langage facilitant leur programmation.

Une procédure d'entrée-sortie regroupe un ensemble "logique" de données en saisie-édition. Un critère de regroupement pourra être : pas d'autres traitements que ceux directements liés au dialogue pendant la saisie-édition de l'ensemble.

Le langage doit fournir deux types de "traitement lié au dialogue"
- les instructions permettant l'indition-édition des données
- les assertions de cohérence

L'indition-édition permet le passage d'une représentation externe habituelle à l'utilisateur à une représentation interne permettant la manipulation par le programme : passage d'une heure à l'équivalent en secondes par exemple.

(heures, minutes, secondes) $\underset{\text{édition}}{\overset{\text{indition}}{\rightleftarrows}}$ nombre de secondes

Les possibilités arithmétiques et logiques classiques suffisent à cette manipulation.

Les assertions de cohérence déterminent le domaine de validité des données échangées. Ces données, particulièrement en saisie, ne sont pas fiables. Un contrôle syntaxique ne peut suffir à assurer leur validité. Le programme va leur associer une certaine sémantique : un entier représentera par exemple le salaire mensuel ... Il est naturel de contrôler que ces données obéissent aux règles sémantiques que l'analyse a formulées : corrélation entre salaire et heures de présence ... Le programmeur détermine ce contrôle sémantique par l'énoncé d'une assertion de cohérence. Ces assertions sont souvent de type ensembliste : le domaine de validité se traduit par un ensemble donné. On peut distinguer la cohérence intrinsèque de la cohérence d'ensemble. La cohérence intrinsèque d'une donnée ne fait pas intervenir d'autres données en saisie. C'est, par exemple, la nécessité de l'existence d'un enregistrement

préalablement à toute mise à jour. La cohérence d'ensemble lie logiquement plusieurs données. Elle ne pourra être évaluée que lorsque la dernière donnée aura été saisie. En cas d'assertion non vérifiée, la donnée invalide n'est pas forcément celle-ci ; il faudra alors permettre une reprise sur une des données de l'ensemble de cohérence.

En plus des deux types de traitement associé au dialogue, le langage doit disposer des structures de données habituelles permettant de définir le type des données manipulées. Parmi celles-ci, les données échangées peuvent être en saisie, en édition, ou en édition-saisie (affichage en vue d'une mise à jour). Les caractéristiques d'affichage (cadres de saisie associés à chaque données) sont définies interactivement grâce à un éditeur de grille : cette mise en écran ne fait donc pas partie du langage de dialogue.

Les structures de contrôles habituelles vont permettre de définir le déroulement normal du dialogue. L'instruction CAS, par exemple, est bien adaptée au choix par "menu" ... Le vrai problème de la programmation en environnement conversationnel vient des déroutements de l'exécution "normale" : les demandes d'aide de l'utilisateur et les reprises en cas de panne ou d'erreur. Ces interruptions intempestives sont très lourdes à définir explicitement. Nous verrons comment la structure de bloc du langage va déterminer les moments d'affichage et d'effacement des cadres de rubriques, ainsi que les possibilités de reprises et la gestion des aides visibles à un moment donné.

3.5 - Structure de blocs et affichage des rubriques

La structure de bloc permet de hiérarchiser les différentes parties d'un programme. Le niveau hiérarchique le plus élevé est bien sûr le programme (bloc englobant). Les blocs englobés affinent le traitement : l'entrée dans un bloc englobé consiste donc à passer du général vers le particulier.

Deux blocs de même niveau englobés par le même bloc sont appelés "blocs jumeaux", et traitent donc deux aspects différents de l'idée plus générale contenue dans ce bloc englobant. Classiquement, la portée d'une variable déclarée dans un bloc s'étend jusqu'à la fin de ce bloc. Pour les données échangées, peut-on lier portée et affichage ? L'utilisateur verrait donc d'abord les rubriques d'ordre général, puis les rubriques correspondant aux cas particuliers de l'exécution. A tout moment, il aurait sous les yeux, sauf

limitations dûes à la taille de l'écran, l'ensemble des rubriques affichées dans les blocs englobants, soit la partie du contexte courant concernant le dialogue (la "grille" courante). Ce principe général semble satisfaisant. Il faut en étudier les conséquences sur les différents types de données échangées : variables en saisie, en édition ou en édition-saisie.

Les variables qui reçoivent leur valeur par saisie n'ont pas de signification avant cette saisie. Il est donc naturel d'associer déclaration et demande de saisie ;

Saisie A entier correspond donc à la déclaration de A et à l'affichage du cadre de saisie associé.

La variable A restera visible jusqu'à la fin du bloc courant. On peut associer affichage et portée des variables en saisie. Le cadre de saisie associé à A s'efface donc lorsque l'on sort du bloc courant

Procédure :	Affichage :
début	( )
saisie A entier	(A)
début	
saisie B entier	(A_B)
fin	(A)
fin	( )

Ce sont donc toujours les derniers cadres affichés qui s'effaceront en premier (blocs les plus internes).

Pour les données en édition ou saisie-édition, on ne peut plus associer portée des variables et affichage. Ces variables doivent être déclarées préalablement à leur affichage pour permettre la recherche de leur valeur (consultation d'une base de données ...). Il en est de même pour les paramètres effectifs faisant l'objet de saisie.

La dissociation déclaration-instruction d'affichage entraîne une distorsion dans le mécanisme d'affichage. La portée de la variable peut excéder le bloc englobant l'instruction d'affichage. Faut-il effacer le cadre d'écran à la fin de ce bloc ou en fin de portée ?

Ces deux stratégies présentent de sérieux inconvénients :
- L'effacement en fin de bloc englobant l'instruction d'affichage prive l'utilisateur d'un contexte encore utile. On verra par la suite qu'il limite ses possibilités de reprise d'une manière peu acceptable.

- L'effacement en fin de portée a des conséquences inacceptables : l'ordre d'effacement ne sera plus obligatoirement l'inverse de l'ordre d'apparition ; les rubriques disparaissant de l'écran ne seront plus forcément les dernières. Pas exemple, le programmeur d'une procédure n'a pas connaissance de l'ordre de déclaration des paramètres par le programme appelant, et cet ordre peut varier d'un appelant à l'autre.

Enfin, lorsqu'une variable en saisie a été utilisée pour l'affectation d'une autre variable (pour faciliter l'indition ...), il peut être utile de garder son cadre affiché après la sortie de sa portée.

On peut donc fournir au programmeur un mécanisme lui laissant le choix entre une "connexion" faible ou forte entre un bloc (ou une procédure) et le bloc qui l'englobe.

La connexion faible (effacement des rubriques internes en fin de bloc ou de procédure) sera implicite. Par contre, une connexion forte (affichage conservé) sera indiquée par le programmeur : on appellera "rémanent" tout bloc (ou procédure) se terminant sans effacement des rubriques internes. Ce terme est employé dans d'autres circonstances avec un sens très voisin :
- un écran rémanent est un écran conservant l'affichage sans rafraichissement, donc ne nécessitant pas de mémoire d'écran
- les données rémanentes d'une procédure sont conservées d'un appel sur l'autre. En dehors de ces appels, bien qu'elles ne soient pas accessibles, elles gardent leurs valeurs qui pourront être ré-utilisées.

Le programmeur disposera donc de deux "fin de bloc" :
- le fin de bloc (ou retour) habituel
- le fin de bloc (ou retour) rémanent.

Cette distinction n'aura aucune influence en dehors du dialogue (même portée des variables ...).

L'affichage des rubriques d'un bloc rémanent se conservera donc jusqu'à la sortie d'un bloc non rémanent englobant.

Grâce à ce mécanisme, le programmeur pourra élargir le domaine d'affichage des cadres de saisie. Cet élargissement concerne l'ensemble des cadres d'un bloc, car l'élargissement sélectif reposerait le problème de l'ordre d'affichage et d'effacement. De plus, la notion de bloc correspond bien à un regroupement logique de données à traiter globalement.

Exemple de procédure de dialogue (cet exemple est développé en annexe) :
Cet exemple est repris de [CARRE 80]. Une entreprise textile dispose d'un fichier de ses fabrications. Un produit est défini par un modèle et par la nature de trois matériaux utilisés. La procédure suivante assure la mise à jour du fichier.

```
type  transaction : ensemble (CREATION, MODIFICATION, SUPPRESSION)
type  modèle      : alphanumérique
type  matériau    : alphanumérique
      début  saisir  (trans : transaction) ;
             cas  trans  dans
          CREATION : répéter jusqu'à entrée vide
                     début  saisir (mod : modèle) ;
                            saisir (paramètres du modèle) ;
                            contrôler (modèle ∉ liste modèles) ;
                            contrôler (contraintes sur les paramètres) ;
                            mémorisation du modèle ;
                            répéter jusqu'à entrée vide
                            début Pour i=1 à 3 début saisie (matériau) ;
                                                     spécif matériau existe
                                                fin rémanent
                                  allouer espace fichier :
                                  saisie (paramètres) ;
                                  memorisation
                            fin rémanent
                     fin (valider)
          MODIFICATION : début ... fin
          SUPPRESSION  : début ... fin
      Fin
```

Remarques :
- les 3 codes "matériau" auraient pu être saisis d'une manière groupée. Les contrôles de respect de spécification auraient alors été faits globalement et non plus matériau par matériau. Mais l'écriture du programme en aurait été simplifiée (saisir [1..3] de matériau))
- l'entrée vide est désignée comme marqueur de fin de liste.

Après avoir défini la structure de l'application, le programmeur déterminera les messages associés à l'aide de l'éditeur interactif.

Variables ou Spécifications (messages par défaut)	Message Associé	Fenêtre Associée (par défaut. Elle peut être augmentée)
trans : transaction	"Type de transaction ?"	Suffisante pour contenir le menu
mod : modèle	"Code du nouveau modèle ?"	
modèle ∉ liste modèles	"Modèle non existant"	ligne système
paramètres	"Paramètres du modèle" mod	

Remarques :
- le programmeur a défini un message dynamique utilisant la variable mod
- il a choisi l'option par défaut pour la variable trans qui est d'un type ensemble (affichage des éléments du type)

Les mécanismes de reprise et d'aide vont dépendre de la structuration en blocs du programme.

3.6 - Les déroutements liés au conversationnel

On distinguera les reprises pour modification de données des aides proposées à l'utilisateur.

3.6.1 - Les reprises

Il faut d'abord définir les possibilités de reprise dont disposera l'utilisateur, et ensuite en étudier les implications sur le langage d'écriture de dialogue.

3.6.1.1 - Possibilités de reprise offertes à l'utilisateur

Le principe général sera le suivant : toute saisie encore affichée peut faire l'objet d'une reprise spécifique. Toutes les saisies antérieures ne devront donc pas être remises en question par cette reprise.

L'utilisateur utilise les touches "Tabulation avant et arrière" pour remonter dans les cadres précédents et se positionner sur le cadre désiré.

La taille de l'écran virtuel (ensemble des données affichées à un moment donné) pouvant dépasser largement la taille de l'écran réel, les derniers cadres pourront disparaître (défilement en rouleau). Tant que l'utilisateur ne modifie pas de données, aucune reprise n'est déclenchée. Ceci lui permet de revisualiser un contexte plus ancien ayant disparu de l'écran faute de place.

Dès qu'une donnée est modifiée, une reprise est déclenchée. Deux stratégies sont alors possibles :
- toutes les saisies postérieures à la donnée reprise disparaissent
- seules les saisies "n'ayant plus de sens" disparaissent.

La deuxième solution est évidemment plus favorable à l'utilisateur. Mais son implémentation risque de poser quelques problèmes : comment différencier les données obsolètes des données à conserver ? La différentiation reprise sur une donnée de définition/reprise sur une donnée de traitement peut-elle suffire ? Ceci sera évoqué dans les extensions possibles à l'implémentation, au chapitre suivant.

3.6.1.2 - Les reprises et le langage

Le programmeur n'a pas à écrire explicitement les possibilités de reprise. Toutefois,l'écriture (la structure) d'un programme a une double influence sur ces possibilités de reprises :
- par l'imbrication des blocs et leurs caractéristiques (rémanents ou non).
- par le groupement en lots de saisie.

On peut donc dire que le programmeur définit implicitement les possibilités de reprise. Il peut grouper plusieurs saisies ou saisies-édition dans la même instruction. Cela donnera lieu à l'affichage d'une sous-grille. Aucun traitement interne à la sous-grille ne pouvant être effectué, toute modification d'une donnée de la sous-grille courante ne nécessite pas de reprise. La transmission d'une sous-grille vers le programme se faisant globalement après sa "validation" par l'utilisateur,les reprises internes sont complètement transparentes vis à vis du programme.

En élargissant le "domaine d'affichage" des rubriques grâce à la rémanence des blocs englobants, le programmeur augmente les possibilités de reprise. L'implémentation devra veiller à ce que cette rémanence ne modifie pas la règle des reprises : tout ce qui est affiché peut être repris, et les données antérieures au lieu de reprise n'en sont pas affectées.

Les possibilités de reprise seront limitées par la libération des données partagées (validation).

3.6.1.3 - Rémanence et validation

Aucune des saisies précédant un point de validation ne pourra rester rémanente après l'exécution de celui-ci, puisqu'elle ne pourra plus être modifiée.

Lorsqu'une application travaille sur des données partagées, elle doit auparavant les verrouiller puis les libérer en fin de traitement. En effet, ces données possèdent une cohérence, c'est à dire vérifient un ensemble de relations et de spécifications. Un traitement particulier peut nécessiter le passage par un état incohérent. Les données momentanément incohérentes sont alors interdites d'accès pour les autres utilisateurs (verrouillage). La libération n'est possible qu'après le retour à un état cohérent.

Passé ce "déverrouillage", toute modification nécessite une "transaction inverse" (réservation ...) qui peut ne plus être possible. L'annulation pure et simple du traitement risquerait de provoquer l'"effet domino" ; annulation en cascade d'autres traitements effectués par d'autres utilisateurs.

On imposera au programme d'associer toute validation à une fin de bloc, ce qui respecte bien la sémantique de la structuration. La validation va impliquer certaines contraintes sur la rémanence.
- un bloc ne pourra pas être en même temps validé et rémanent.
- un bloc validé ne pourra pas être englobé par un bloc rémanent
- les données échangées avant le bloc validé ne seront plus modifiables après, même si l'on reste dans leur portée.

```
exemple :  début
                saisie (A)
                          début
                          fin validé ;
                instructions
           fin
```

A n'est plus modifiable par l'utilisateur pendant l'exécution des instructions.

La validation détruit donc la structuration en blocs : un bloc validé affecte les blocs englobants en cours. Pour l'utilisateur, il serait préférable de reculer toute validation en fin du bloc le plus externe. Mais ceci implique

alors un verrouillage long, qui gênera peut-être d'autres utilisateurs (mise en file d'attente ...). On retrouve le problème des "parts de gâteau" : lorsque plusieurs utilisateurs se partagent une ressource critique, l'avantage donné à l'un deux (ici, possibilité de reprises) l'est au détriment des autres (indisponibilité des données).

Ce sera donc au programmeur de choisir le compromis qui lui semblera le plus acceptable.

3.6.1.4 - Possibilités de validation implicite

Dans certains cas précis d'application, il est possible de déduire le lieu de validation de la portée de déclaration des données externes et de la rémanence des blocs. La solution proposée pour le mécanisme de reprise (tout ce qui est visible par l'utilisateur peut être repris) interdit les validations pendant la rémanence (paragraphe précédent). On peut alors convenir d'associer automatiquement une validation au bloc le plus englobant contenant une déclaration de donnée externe (réservation) et non englobé par un bloc rémanent (mécanisme déjà évoqué dans [CARRE 80] : voir 2.10).

Mais d'autres types de contraintes peuvent être envisagés.

On se trouve en présence de trois notions différentes ayant des implications réciproques :

- lieu de déclaration des données externes : ces données sont, a priori, verrouillées pendant toute leur portée,
- rémanence : conserve la visualisation au-delà de la portée, ce qui, dans le mécanisme adopté, conserve les possibilités de reprise
- validation : interdit toute reprise sur le traitement antérieur, mais n'implique pas forcément un déverrouillage total.

D'autres hypothèses de contraintes mutuelles pourraient modifier le mécanisme de reprise :

- tout bloc est soit rémanent soit validant : ceci interdit les blocs non rémanents sans validation, qui permettent de supprimer certaines données du contexte tout en autorisant des retours sur des données antérieures. On pourra toujours limiter le contexte visible du bloc en lui associant une fenêtre minimale, mais en cas de retour par "TABULATION ARRIERE", toutes les données du bloc seront revisualisées.

- Déclaration de données externes dans des blocs englobants des blocs validants : le maintien des verrous n'est plus justifié par les possibilités de reprise mais parce qu'il évite l'attente de données partagées utilisées par plusieurs blocs validants consécutifs (concept de transactions emboîtées étudié dans [FRANK 82])
- possibilité de créer des blocs rémanents englobants des blocs validants, ce qui permet de visualiser un contexte déjà validé. Il faut alors une "astuce de visualisation" (changement d'intensité ...) pour différencier les données modifiables des autres.

Toutes les combinaisons entre ces contraintes et même l'absence de contraintes sont envisageables. Suivant le type d'application, ceci pourra être laissé à l'appréciation du programmeur ou au contraire imposé par le compilateur.

Le mécanisme de reprise proposé a donc deux avantages : être simple pour l'utilisateur (toute saisie encore affichée est modifiable) et ne nécessitant pas une programmation fastidieuse (utilisation de la structure des blocs). Ceci sera bien sûr payé par une plus grande complexité d'implémentation du langage.

L'autre famille de déroutements "spéciaux" concerne les demandes d'aide par l'utilisateur.

3.6.2 - *Définition et mécanisme d'apparition des aides-mémoires*

Par souci de concision des cadres de rubriques et de grilles, les messages ne pourront pas spécifier entièrement le déroulement du programme. L'utilisateur doit disposer d'une touche "aide" permettant l'affichage de compléments d'explications. Cet affichage peut être automatique en cas d'erreurs successives. Ce complément d'information devrait être "incrémental" et dépendant du profil-utilisateur.

L'aide proposée peut concerner divers niveaux :
- la syntaxe imposée par le programme
- les restrictions sur les données et leurs liaisons
- le point courant du programme.

Un message expliquant la syntaxe peut être associé à chaque type de donnée. Le programmeur n'aura à le spécifier qu'à la déclaration du type.

Les assertions de cohérence pourront figurer comme aide ou messages d'erreur plus "sémantiques". Mais leur énoncé risque d'être déroutant pour l'utilisateur, aussi le programmeur pourra leur associer un message en clair.

Enfin, le programmeur peut associer à chaque bloc un texte spécifiant le rôle du bloc. En cas de demande d'aide, on affichera le texte du bloc courant, un nouvel appui de la touche aide permettant d'afficher les textes des blocs les plus englobant.

Cet affichage ira donc du particulier (point courant du programme) au général (explication du bloc le plus englobant, c'est à dire du programme).

Si le système gère un profil utilisateur (débutant - confirmé ...) le programmeur pourra éventuellement donner plusieurs versions de ces textes. La définition de ces messages peut être incorporée au programme lui-même (programme auto-documenté) ou se faire ultérieurement à l'aide de l'éditeur d'écran (ce qui facilite sans doute la tâche du programmeur). On pourra toujours éditer un listing du programme en y incorporant les messages.

La gestion automatique des déroutements liés au conversationnel permet donc de concilier la souplesse du dialogue et la facilité de programmation.

3.7 - Exemple de déroulement d'un dialogue

Voici une exécution possible de l'application proposée au paragraphe 5 :

TYPE DE TRANSACTION : CREATION
MODIFICATION
SUPPRESSION

CODE DU NOUVEAU MODELE : blouson

PARAMETRES DU MODELE : tailles enfant ...

NOUVEAU PRODUIT
MATERIAU coton
MATERIAU cuir
MATERIAU fermeture glissière
PARAMETRES DU PRODUIT couleurs noir/rouge

NOUVEAU PRODUIT
MATERIAU ...
MATERIAU
MATERIAU
PARAMETRES DU PRODUIT

NOUVEAU PRODUIT
MATERIAU ...
MATERIAU
MATERIAU
PARAMETRES DU PRODUIT

NOUVEAU PRODUIT : |_|

Remarques :
- l'utilisation de codes augmenterait la vitesse de saisie (on peut afficher l'élément correspondant à titre informatif).
- des messages d'erreurs auront pu être affichés sur la ligne système. L'utilisateur aura alors corrigé les données erronées.
- l'ensemble des rubriques est virtuellement affiché : ne sera réellement affiché que ce qui tient sur l'écran (disparition des rubriques du haut de l'écran). Mais l'utilisateur peut se servir des touches de tabulation pour revoir les rubriques disparues
- la rémanence s'arrête au niveau du modèle
- quand l'utilisateur répondra à la question NOUVEAU PRODUIT ? par l'entrée vide, une demande de confirmation apparaitra sur la ligne système. En cas de validation, le traitement devient effectif et il n'y a plus possibilité

de modifier une de ces saisies
- par contre, avant ce point de validation, toutes les données peuvent être modifiées par l'utilisateur (y compris les données du modèle).

La liste des modèles créés (ressource partagée) sera a priori verrouillée. Mais on peut permettre la lecture seule par d'autres utilisateurs. Ils n'auront pas alors connaissance du modèle en cours de création.
- En cas de modification d'une donnée, les saisies qui la suivent sont perdues. Une extension du mécanisme, permettant de conserver les données encore valides, sera proposée au chapitre suivant.
- Enfin, la possibilité d'associer une fenêtre aux blocs de programme permet d'éviter l'affichage "in extenso" des produits, ce qui conserve l'affichage sur l'écran de contexte du modèle. Cette possibilité n'a pas été utilisée ici.

Par rapport aux logiciels habituels, une très grande liberté de manoeuvre est donc laissée à l'utilisateur.

3.8 - Apport du langage par rapport aux logiciels existants

Ce langage étant défini (informellement), il est intéressant de lui appliquer les critères de comparaison des logiciels exposés en bibliographie.

L'effort de programmation est réduit à l'essentiel :
- structurer convenablement l'application
- définir les structures de données, leurs types ...
- définir les contraintes sémantiques sur ces données.

Hors, ces trois points relèvent plus du domaine de l'analyse que de la programmation. Ce langage respecte donc la tendance de la décennie : inclure des concepts du niveau d'analyse (modules ...) pour permettre des applications plus fiables et limiter le travail de "transcodage".

La définition précise de la mise en écran est reportée à un stade ultérieur et grandement facilitée par un éditeur interactif (messages définis et cadrés directement sur écran). Un maximum d'options par défaut permet même une mise au point du programme sans définition soignée du dialogue. C'est la pratique habituelle, mais trop souvent, la définition précise de ce dialogue oblige alors à se replonger dans le texte du programme lui-même. L'éditeur d'écran facilitera la modification des messages et permettra d'adapter un dialogue à une taille d'écran particulière.

La mise en écran des grilles en table est simple (défilement en rouleau dans une fenêtre). Ce principe est généralisé aux blocs du programme, ce qui permet au programmeur de choisir le contexte réellement visualisé.

La description de l'enchaînement des "grilles" est implicite : elle se déduit de la structure de l'application.

A l'exécution, une grande souplesse de manoeuvre est laissée à l'utilisateur, tout en assurant un contrôle rigoureux.

Les contrôles syntaxiques sont déduits du type des variables saisies. Les contrôles plus sémantiques (liaisons entre les données ...) sont définis explicitement par les assertions de cohérence. Leur énoncé "procédural" entraine une restriction sur l'affichage des sous-grilles : pour pouvoir afficher simultanément plusieurs sous-grilles, il faudra retarder leurs vérifications.

Un mécanisme simple permet à l'utilisateur de modifier une donnée acceptée par le programme : il lui suffit de repositionner le curseur au bon endroit. De plus, il aura à sa disposition des messages d'aides adaptés au point d'exécution courant.

Seule la nécessité de libérer des données partagées impose certaines restrictions à l'utilisateur.

On pourra retrouver le schéma transactionnel habituel (affichage-saisie/vérification-traitement/libération-effacement), mais en beaucoup plus souple, en rendant rémanents tous les blocs internes à un bloc validé, qui sera alors une transaction (pouvant être complexe).

Mais une utilisation moins systématique de la rémanence aura sans doute sa place dans d'autres applications.

Il faut maintenant étudier les implications de ces spécifications sur une implémentation du langage.

4 - QUELQUES VOIES D'IMPLÉMENTATION

L'implémentation du langage défini au chapitre précédent devra résoudre les problèmes suivants :

- fonctionnement de l'éditeur de cadres d'écran par rapport au texte source des procédures de dialogue et à l'archivage des rubriques déjà créées.
- communications procédures de dialogue/grilles d'écran
- gestion de l'écran virtuel : ajouts et suppressions de sous-grilles dans le contexte d'affichage, position de l'écran réel, gestion des remontées du curseur dans l'écran virtuel ...
- implémentation des points de reprise
- communications procédures de dialogue/langage de traitement

Le fonctionnement de l'éditeur interactif de cadres d'écran ne pose aucun problème majeur. En analysant le texte source des procédures de dialogue, il doit retrouver les rubriques de saisie non encore définies (gestion d'un fichier des rubriques) et les construire en demandant au programmeur les messages et les fenêtres associés lorsqu'il ne désire pas les options par défaut. Le code représentant chaque structure de saisie dépend du terminal utilisé : une partie de l'éditeur sera donc très liée au type de matériel.

L'écran virtuel sera géré à l'aide d'une structure de donnée appropriée (arborescence) permettant de tenir compte des blocs virtuellement affichés ainsi que des fenêtres réellement affichées.

Une seconde structure de donnée, la trace du dialogue, conservera l'ensemble des données en saisie ou en édition-saisie, et permettra la "transparence" de certaines reprises.

Les points de reprises seront gérés en pile, et le traitement de reprise associé pourra être défini par un traitement d'exception de type ADA.

Les communications dialogue/traitement seront fortement dépendantes des possibilités de paramétrage offertes par le langage de traitement. L'utilisation du mécanisme de rendez-vous de type ADA pourrait exploiter les possibilités de parallélisme dialogue/traitement : saisie par anticipation ...

Ce chapitre s'attachera surtout à montrer la faisabilité d'une gestion non explicitement programmée des reprises. En tenant compte d'un parcours très particulier des arborescences nécessaires, une implémentation "minimale" sera proposée. Le dernier paragraphe discutera d'une extension du mécanisme au profit de l'utilisateur par l'exploitation de la structure arborescente d'une trace plus complète.

4.1 - Les reprises

4.1.1 - Reprises et restitutions du contexte

Une reprise sur une donnée en saisie peut être déclenchée soit par le système informatique, soit par l'utilisateur. Les reprises système correspondent par exemple à une assertion de cohérence d'ensemble non vérifiée, ou à l'indisponibilité d'une donnée partagée, ou même à une panne ou à un risque d'interblocage. Les reprises utilisateurs correspondent à des données acceptées par le programme mais erronées du point de vue de l'utilisateur.

Provoquer une reprise ne consiste pas simplement à dérouter le programme de son exécution normale. Il faut restituer le contexte initial du lieu de reprise, c'est à dire redonner aux objets manipulés par le programme leur valeur courante au moment de la saisie qui vient d'être modifiée.

On appelle point de reprise l'endroit du programme où le contexte est sauvegardé. A chaque point de reprise est associé un bloc de reprise, qui est la région du programme où la reprise sur ce point est permise. Il parait naturel de créer une hiérarchie de points de reprise en imbriquant les blocs de reprise : ceci revient à créer une arborescence des points de reprise que l'on gère en pile.

Deux classes d'objets sont à considérer du point de vue de la restitution de contexte : les objets internes au programme et les objets externes.

Pour restituer le contexte interne, il suffit de mémoriser la valeur des variables à l'entrée du bloc de reprise. Les variables internes au bloc ne sont donc pas à considérer. En fin normale du bloc, la copie est détruite. En cas de reprise, elle permet la restitution du contexte initial. Pour éviter une trop grande perte de place mémoire, il suffit de ne recopier que les données modifiées à l'intérieur du bloc.

Le contexte externe pose le problème de partage des objets. Les fichiers et bases de données sont accessibles "simultanément" par plusieurs utilisateurs. Ceci implique une synchronisation : les objets manipulés par un utilisateur deviennent inaccessibles pour les autres.

Deux aspects de la restitution du contexte externe sont à envisager :
- la restitution des valeurs initiales des enregistrements : suivant le type de matériel et d'application, les mises à jours peuvent se faire sur des copies ou être retardées jusqu'à un point de validation (mécanisme de liste d'intentions [FRANK 82])
- la libération des enregistrements réservés : ce sera le rôle d'une routine spécifique de reprise. Il ne suffit pas en effet de conserver les anciens paramètres de gestion des fichiers externes, car ces paramètres ont pu être modifiés depuis par d'autres utilisateurs.

Il faut donc créer les "mouvements inverses".

Peut-on identifier bloc de reprise et bloc de programme contenant des saisies ?

4.1.2 - Les blocs de reprises

On permet à l'utilisateur de reprendre toute donnée encore (virtuellement) affichée. Cependant la reprise sur une donnée ne devrait pas conduire à la re-saisie de données qui la précédent. L'identification bloc de reprise/bloc du programme contenant des saisies respecte bien la sémantique du dialogue mais ne satisfait pas cette contrainte.

4.1.2.1 - Identification bloc de reprise et de programme

La création d'un bloc par le programmeur correspond au traitement d'un "niveau" de l'algorithme. L'imbrication des blocs permet de passer d'un traitement plus général au traitement d'un point particulier. L'ensemble des données échangées entre le système et l'utilisateur au même niveau du traitement (même bloc) peut donc être mise au même "niveau sémantique".

La création d'un point de reprise à chaque ouverture de bloc de programme respecte bien cette sémantique. Lors de la demande de reprise sur une donnée du bloc, l'utilisateur se replace sur le début de l'"ensemble sémantique" formé par le bloc et ses données. Mais ceci est insuffisant pour implémenter le principe de reprise défini au chapitre précédent, puisqu'on oblige l'utilisateur

à re-saisir toutes les données du bloc, même s'il ne voulait pas modifier les premières. Cette contrainte, très désagréable, risquerait même de dissuader l'utilisateur non averti de l'utilisation des possibilités de reprise. Ne dominant pas clairement la structure de l'application, il ne pourrait prévoir la "profondeur" du retour-arrière qu'on exigerait de lui.

Toute valeur échangée entre le début du bloc et la donnée modifiée par l'utilisateur reste inchangée. La mémorisation des valeurs échangées, qui est différente de la sauvegarde du contexte, permet d'éviter les re-saisies fastidieuses. Ceci se fait implicitement lors d'une modification d'une donnée dans une saisie groupée non terminée. Dans le langage proposé, comme dans tous les logiciels de grilles d'écran, on utilise alors la mémoire d'écran comme trace. Ceci est évidemment insuffisant dès que l'on sort de la saisie groupée courante. Une trace (mémorisation des valeurs échangées), va alors rendre la ré-exécution du début du bloc transparente à l'utilisateur. Mais ceci présente deux inconvénients :
- la réexécution du début de bloc peut allonger considérablement le temps de réponse aux reprises
- en cas d'utilisation de données partagées, les données libérées par le traitement de reprise ne pourront peut-être plus être réservées à nouveau.

La création d'un point de reprise par instruction de saisie résoud ce problème mais détruit, dans certains cas, la structuration du programme.

4.1.2.2 - Création d'un point de reprise à chaque saisie

On ouvre un bloc de reprise devant chaque saisie. Quelle portée doit-on donner à ce bloc de reprise ? L'idéal serait d'étendre ce bloc jusqu'au moment où une reprise spécifique sur la donnée n'est plus possible. Or, une reprise reste possible tant que la donnée est affichée. Le point d'effacement de la structure de saisie associée est la première fermeture d'un bloc englobant non rémanent. Les rubriques d'un bloc rémanent restent en effet affichées après la fermeture du bloc (voir chapitre 3). L'extension du bloc de reprise jusqu'à ce point peut entrainer le non-respect du principe d'imbrication des blocs.

Exemple :

```
début 1   ...   début 2  saisie A  ;   ...
                fin 2 rémanent ;

          ...

fin 1
```

Le bloc de reprise associé à A chevaucherait les blocs 1 et 2.

Cela reviendrait à provoquer une reprise interne à un bloc (ou une procédure) déjà terminé. La gestion du contexte nécessaire aux reprises ne pourrait plus s'appuyer sur la structuration du programme et la portée des variables. Ceci n'est donc pas acceptable. Les blocs de reprises, créés devant chaque saisie, se termineront donc avec le bloc immédiatement englobant, qu'il soit rémanent ou non.

Si le point d'exécution courant au moment de la demande de reprise sur une donnée est à l'intérieur du bloc de programme contenant l'instruction de saisie, la reprise peut se faire directement sur cette saisie : le bloc de reprise courant est le bloc de reprise associé à la donnée.
Exemple de reprise sur une saisie d'un bloc englobant :

```
début 1
        début 2  saisie (A) ...  fin 2  ...
                              ↑
                  point d'exécution courant
fin 1
```

Par contre, si le bloc contenant la saisie est déjà terminé (bloc forcément rémanent), le bloc de reprise à utiliser englobera ce bloc terminé. Le lieu de reprise sera donc forcément "en amont" de la donnée modifiée.

Reprise sur une saisie d'un bloc terminé :

```
début 1
        début 2  saisie (A)  ...  fin 2 rémanent  ...
                                                   ↑
                                        point d'exécution courant.
fin 1
```

Il est donc utile de créer un point de reprise juste avant tout bloc rémanent. La mémorisation des données échangées permettra de se replacer sur la saisie modifiée sans intervention de l'utilisateur (transparence de la réexécution).

La reprise à l'intérieur d'un bloc non rémanent terminé n'est pas possible : toutes ses rubriques de saisie sont effacées. Mais ceci n'implique pas l'impossibilité de modifier les données échangées dans ce bloc. En effet, des données antérieures au bloc non rémanent terminé peuvent encore être affichées :

```
début   ...
            saisie (A) :
        ...
        début saisie (B)    fin
        ... ← la rubrique associée à A est encore affichée
fin
```

En cas de reprise sur une de ces données, il faudra re-saisir les données du bloc non rémanent terminé. La notion de rémanence de bloc est donc différente de la notion de validation ; aucune donnée antérieure à une validation ne peut être reprise passé ce point de validation.

La création d'un point de reprise par instruction de saisie entraîne donc une structuration supplémentaire imbriquée dans la structuration imposée par le programmeur.

Exemple : (DR - FR délimitent un bloc de reprise) :

```
Programme "source"              Blocs ajoutés
début ...                       début ...
      saisie (A) ;                    DR1 saisie (A) ;
      ...                             ...
      saisie (B) ;                    DR2 saisie (B) ;
      ...                             ...
                                      FR2 ; FR1
fin                             fin
```

On obtiendrait le même résultat en imposant au programmeur la structure suivante pour les blocs contenant des saisies

```
début  saisie (X)   instructions  fin
```

et en y associant un bloc de reprise : on appellera bloc de saisie un bloc possédant cette structure. Cette contrainte n'est pas totalement arbitraire ; la saisie d'une valeur (qui n'est donc pas un résultat immédiat de ce qui précéde) correspond bien à l'entrée dans un "niveau sémantique" différent (un nouveau bloc). Mais puisque cette structuration peut se faire a posteriori, il ne parait pas utile de l'imposer au programmeur.

Le principe de reprise offert à l'utilisateur (toute donnée affichée est modifiable sans re-saisie de données antérieures à celle-ci) peut donc

s'implémenter grâce au double mécanisme :
- création, à la compilation, de blocs de saisie
- gestion, à l'exécution, d'une trace de dialogue (ensemble des valeurs échangées).

La sauvegarde de la trace du dialogue permet d'éviter de créer un point de reprise par instruction de saisie. A la limite, un point de reprise après chaque point de validation peut paraitre suffisant. On retombe alors dans le schéma classique des transactions "indivisibles": saisie/validation /traitement. Il faut cependant rappeler que la non-création de certains blocs de saisie peut être dangereux :
- soit parce qu'elle rallongerait le temps de réponse aux reprises
- soit, et c'est beaucoup plus grave, parce que le traitement de reprise libérerait des données partagées qui ne pourrait peut-être plus être acquises à nouveau.

Cette possibilité de se passer de la création de certains blocs de saisie peut néanmoins se révéler intéressante dans les cas où leur création entrainerait une trop grande perte mémoire (un point de reprise est un point de mémorisation de contexte). Il faut donc étudier l'impact sur ce coût de mémorisation du contexte de cette structuration a posteriori en blocs de saisie.

4.1.3 - Coût de mémorisation du contexte

Ce paragraphe traite de l'influence de la forme des blocs sur la nécessité de mémoriser le contexte. On y montre que la multiplication des blocs de saisie n'entrain pas une mémorisation systématique de tout le contexte.

La mémorisation du contexte d'un bloc consiste à sauvegarder la valeur des variables modifiées dans ce bloc ou dans les blocs englobés. Lorsque ces valeurs concernent la gestion de fichiers partagés un traitement de libération des données réservées pourra y être associé. Or la création d'un bloc de reprise à chaque saisie multiplie les blocs emboîtés.

début

saisie (A) ; instructions 1

saisie (B) ; instructions 2

saisie (C) ; instructions 3

fin

donne

DR1 saisie (A) ; instructions 1

DR2 saisie (B) ; instructions 2

DR3 saisie (C) ; instructions 3 FR3, FR2, FR1.

Pour éviter qu'une variable modifiée par le groupe d'instructions 3 et non modifiée par les groupes 1 et 2 soit mémorisée aux trois points de reprise, on appliquera le principe suivant :

Lorsqu'une reprise est provoquée, on exécute tous les traitements de reprise des blocs de reprises courant dont il faut sortir. Dans l'exemple précédent, une reprise sur B, lors de l'exécution du groupe d'instructions 3 entraine l'exécution des traitements de reprise associés aux blocs 3 et 2. Ceci permet donc la mémorisation la plus "interne" possible. Cette mémorisation unique par le bloc englobé n'est possible que lorsque trois conditions sont vérifiées :

1) Le bloc englobé n'est pas itératif (bloc simple)
2) Le bloc englobé est terminant vis à vis du bloc englobant (pas d'instructions entre les deux "fin de bloc")
3) la donnée n'est pas modifiée en dehors du bloc englobé.

1) Si le bloc englobé est itératif, une double mémorisation de la valeur modifiée permet de dissocier la reprise sur l'itération courante d'une reprise sur une itération précédente :

début

faire plusieurs fois début ... saisie (A) ; modification de valeur (B) fin

fin

donne

début

DR1 faire plusieurs fois début ...DR2 saisie (A).modification de valeur(B)..

FR2 fin

FR1

fin

La valeur de B est mémorisée aux points de reprise 1 et 2.

2) S'il existe des instructions entre les deux "fin de bloc", une double mémorisation est nécessaire :

```
début  ...
       début  ...  saisie (A) ; modification de B fin
       instructions
fin
                              donne
début  DR1 ...
           début ... DR2 saisie (A) ; ... FR2 fin
           instructions
       FR1
fin
```

car le déclenchement d'une reprise sur A lors de l'exécution des instructions provoque une reprise sur le point de reprise 1 sans exécution du traitement de reprise 2 (bloc terminé).

Vis à vis des blocs englobés (créés ou non par le programmeur), les blocs de reprise surajoutés satisfont toujours les conditions 1) et 2). Par construction, ce sont toujours des blocs terminants simples.

Par contre, la dernière règle, liée aux lieux de modifications des variables, n'est pas toujours vérifiée :

```
début  ...
       saisie (B)
       modification de A
       saisie (C)
       modification de A
fin
              donne
début  ...
          DR1  saisie (B)
          ...
          DR2  saisie (C)
          ...
          FR2
          FR1
fin
```

avec une double mémorisation de A.

Si le coût de mémorisation est important (tableaux ...), on peut éviter de créer certains blocs de reprise. Il faudra alors veiller à ce qu'aucune libération par un traitement de reprise de ressources partagées ne puisse bloquer la reprise demandée. Le "positionnement" exact sur la donnée modifiée se fera alors à l'aide de la trace du dialogue. Mais le mécanisme n'étant plus systématique, il faudra trouver un algorithme de choix en précisant le critère "coût de mémorisation important".

La création de blocs de reprises associés à chaque intruction de saisie et à chaque bloc du programme ne multiplie donc pas systématiquement les points de mémorisation des variables.

Mais elle ne résoud qu'en partie le problème de la gestion des reprises. Dans certains cas (blocs rémanents terminés ou coût de mémorisation trop important), on ne peut se repositionner sur le point de reprise précédent immédiatement la donnée que l'on veut modifier. Il faudra donc réexécuter un certain nombre d'instructions, comprises entre le point de reprise et l'instruction de saisie. Or, pour respecter le principe de reprise, cette réexécution doit être complétement transparente à l'utilisateur. En particulier, si ces instructions comportent des saisies, celles-ci ne doivent pas lui être redemandées. La mémorisation des données échangées (qui n'est pas la mémorisation du contexte) va permettre l'exécution de ces saisies sans intervention de l'utilisateur : c'est une généralisation de l'utilisation de la mémoire d'écran lors des reprises internes aux saisies groupées.

4.1.4 - *Trace du dialogue*

La trace du dialogue va donc participer, dans certains cas, au mécanisme de reprise. Cette trace doit contenir toutes les valeurs en saisie ou en édition-saisie échangées depuis le dernier point de validation. Les reprises en-deçà d'un point de validation étant interdites (problème de libération des variables partagées), il n'est pas utile de conserver les valeurs précédentes (sauf peut-être pour visualisation du travail effectué).

La structure de la trace dépend directement du mécanisme de reprise : la reprise se fait en début du plus petit bloc englobant le point courant d'exécution et l'instruction de saisie de la valeur que l'utilisateur vient de modifier.

La structuration de la trace devra donc refléter la structuration en blocs de reprise.

Le terminal fournira au dialogue les renseignements nécessaires à l'identification du point courant d'exécution et de la donnée reprise.

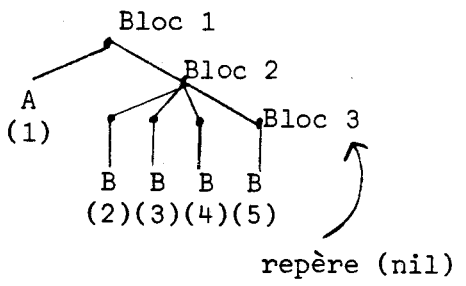
Vis à vis de la trace, le point courant d'exécution est immédiat : c'est la dernière saisie . Les saisies se faisant séquentiellement, un numéro d'ordre pourra être attribué à chaque saisie. Ce numéro,connu du terminal et du dialogue, servira à identifier la donnée modifiée par l'utilisateur qu'il faudra mettre à jour dans la trace. Le principe de reprise défini au troisième chapitre spécifie que les données qui suivent la donnée modifiée ne sont pas réutilisées : toutes ces données seront donc à effacer. La fin du chapitre traitera d'une extension du mécanisme de reprise permettant de conserver les valeurs ayant encore un sens.

Enfin, il faut gérer un repère dans la trace qui donne la prochaine valeur "en saisie" si elle est connue ou qui indique que cette valeur doit réellement être saisie (repère "nil").

Exemple

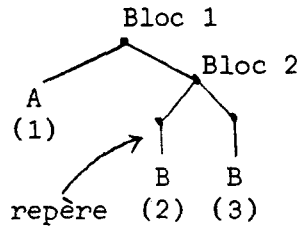
```
Procédure
début
      saisie (A)                                DR1   saisie (A)
      faire plusieurs fois début                  DR2 faire DR3
                                 saisie (B)                    saisie (B)
                           fin                             FR3
fin                                               FR2
                                                  FR1
```

trace d'une exécution



L'utilisateur a modifié B(3) : on reprend sur le bloc de reprise n° 2

trace après le déclenchement de la reprise :



B(3) contient la nouvelle valeur.

En résumé, l'exécution d'une reprise, qu'elle soit provoquée par l'utilisateur ou exigée par le non-respect d'une assertion de cohérence, consiste, vis à vis de la trace à

- identifier la donnée modifiée et la mettre à jour
- supprimer les données qui la suivent
- repérer la première donnée en saisie du bloc repris.

L'identification de la donnée modifiée est réalisée grâce au numéro d'ordre associé à chaque saisie. Le terminal doit donc gérer l'écran de manière à retrouver le numéro d'ordre de la donnée que l'utilisateur modifie. Pour cela, le terminal va construire une structure représentant l'ensemble des données visibles par l'utilisateur à un moment donné : l'écran virtuel.

4.2 - L'Ecran Virtuel

Le terminal est supposé disposer d'une mémoire propre et de possibilités de gestion de l'écran virtuel. Celui-ci représente la grille courante, c'est-à-dire l'ensemble des valeurs échangées encore visibles par l'utilisateur mises en écran dans leurs rubriques associées. Cette grille peut être de taille importante. En particulier, elle n'est pas limitée par la taille de l'écran physique. Il n'est donc pas raisonnable de consacrer à l'écran virtuel une mémoire de type mémoire d'écran, où un caractère de l'écran correspond à un mot mémoire.

L'écran virtuel sera une structure de données contenant les informations nécessaires au remplissage de la mémoire d'écran : valeurs, rubriques associées, fenêtres réellement affichées. Elle contiendra également les données nécessaires

à la localisation des saisies modifiées par l'utilisateur en cas de reprise : numéro d'ordre de chaque saisie. Enfin, la structuration du programme devra apparaitre dans l'écran virtuel.

4.2.1 - Structures de saisie et affichage

Le programmeur a associé à chaque bloc de saisie (ensemble de données ne nécessitant pas de traitements intermédiaires) une structure de saisie, contenant les messages à afficher ainsi que les formats de saisie (type ...). L'écran virtuel contiendra les données échangées (en saisie, édition ou édition-saisie) lors de l'exécution d'une structure de saisie et le moyen d'accéder à la structure de saisie correspondante. Ces informations y sont conservées tant qu'elles sont virtuellement affichées. Il est important de remarquer que, bien que paraissant proches, l'écran virtuel et la trace du dialogue sont deux structures différentes. En particulier, la trace du dialogue n'a pas à contenir les données en édition seule ni le moyen d'accéder aux structures de saisies. De plus, à la sortie d'un bloc non rémanent, les informations internes au bloc disparaissent de l'écran virtuel (elles ne font plus partie de la grille courante) alors qu'elles subsistent dans la trace du dialogue (reprise sur une donnée antérieure toujours possible).

Si le terminal est adapté, des fenêtres graphiques peuvent être affichées. La "structure de saisie" sera alors une primitive ou un ensemble de primitives opérant sur des données mémorisées dans l'écran virtuel.

L'exécution par le terminal d'une instruction saisie (A) consistera à
- chercher et afficher le cadre de grille associé
- effectuer la saisie en contrôlant la syntaxe
- ajouter à l'écran virtuel cette saisie, le nom de la rubrique et le numéro d'ordre de la saisie
- retourner la donnée à la procédure de dialogue

Celle-ci pourra ou non être "résidente" sur le terminal.
Le numéro d'ordre, obtenu par incrémentation d'un compteur, apparait également dans la trace du dialogue. Il permet de localiser la donnée modifiée en cas de reprise.

Pour les structures ne comportant que des données en édition, le mécanisme est beaucoup plus simple : pas de numéro d'ordre, pas de vérification syntaxique.

Lorsque la sous-grille comprend la répétition d'une même structure, l'opération sera effectuée autant de fois que nécessaire, avec transmission en bloc des résultats au dialogue (saisie groupée).

Du point de vue de l'écran virtuel, il est indifférent que ces sous-grilles répétitives soient utilisées par une instruction de saisie plutôt que programmées explicitement.

exemple :
saisie (B : [1 .. n] de A)
pourra être simulé par
Pour i = 1 à n début saisie (A) fin rémanent

La compilation des deux versions donnerait sans doute un code très voisin.

En ce qui concerne l'ensemble du mécanisme d'affichage, on peut toutefois trouver deux différences entre sous-grilles répétitives et structures itératives :
- en cas de reprise interne à une sous-grille dont on n'est pas sorti, la procédure de dialogue n'en aura pas connaissance : les données ne lui ont pas été transmises. Le code permettant l'exécution d'une sous-grille pourra,par exemple, être résident sur le terminal alors que la procédure de dialogue sera exécutée par le processeur central. Au cas où la sous-grille est explicitement programmée (on ne parlera plus de sous-grille), la reprise sera effectivement déclenchée.
- Une sous-grille ne peut contenir des rubriques effacées en même temps que d'autres virtuellement affichées : il ne peut y avoir de structurations fermées internes. Ceci est par contre possible pour les structures itératives explicites. A l'inverse, le langage de dialogue ne prévoit pas d'associer une fenêtre à un bloc de programme, alors que la déclaration d'une sous-grille à plusieurs niveaux (B : [1 .. n] de A et A : [1 .. m] de C) permet d'associer une fenêtre à chaque niveau. Cette distorsion pourrait être supprimée par la possibilité d'associer une fenêtre à un bloc de programme : mais ceci serait en contradiction avec le principe de séparation entre la programmation d'une application et la mise en écran des rubriques et sous-grilles.

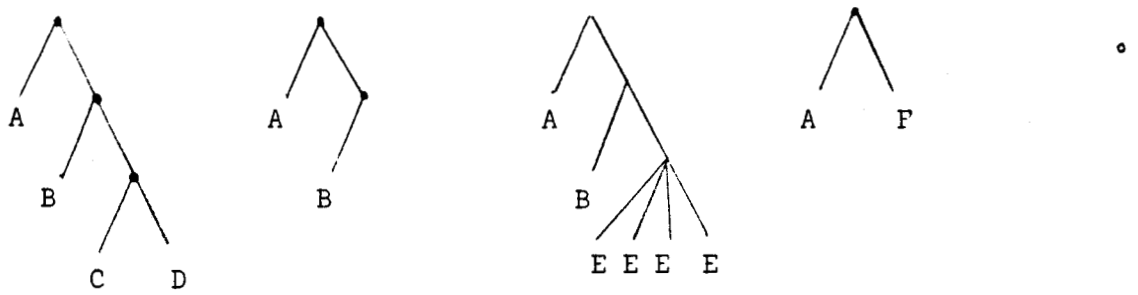
La structure de l'écran virtuel va refléter la structuration en blocs des procédures de dialogue.

4.2.2 - *Structuration de l'écran virtuel*

Pour gérer le mécanisme de reprise, il n'est pas utile que l'écran virtuel soit structuré, comme la trace, par tous les blocs de reprise. Par contre, la structuration de l'écran virtuel doit, au minimum, refléter la structuration du programme en blocs non rémanents. En effet, l'exécution du fin de bloc non rémanent doit entrainer la disparition de toutes les rubriques internes. L'écran virtuel sera donc une structure arborescente (imbrication des blocs) chaque sous arbre représentant un bloc du programme. A la rencontre d'une fin de bloc non rémanent, il suffira donc d'enlever de l'écran virtuel le sous-arbre courant, qui sera toujours sur une "branche extérieure droite" (Convention de construction de haut en bas et de gauche à droite).
exemple :

```
début 1  saisie (A)
         début 2  saisie (B)
                  début 3  saisie (C) ;
                           saisie (D)
                  fin 3 ;
                     faire plusieurs fois début 4 saisie (E)
                                          fin rémanent 4
         fin 2 ;
         saisie (F)
fin 1
```

Evolution de l'écran virtuel



Avant fin 3 Après fin 3 Avant fin 2 saisie (F) fin 1

Remarque :

La création d'un sous arbre par bloc rémanent n'est pas utile à la gestion de l'effacement des rubriques. En effet, du point de vue de l'affichage et de l'effacement des rubriques, les deux programmes suivants sont équivalents.

```
début 1                                    début 1 saisie (A)
      début 2 saisie (A)          et                saisie (B)
      fin 2 rémanent,
fin 1 saisie (B)                           fin 1
```

Mais la structuration de l'écran virtuel par tous les blocs du programme (rémanents ou non) permettra d'éviter de gérer différemment les sous-grilles répétitives et les structures itératives du dialogue.

La trace du dialogue et l'écran virtuel sont donc deux structures arborescentes contenant des données échangées. Mais elles sont fondamentalement différentes :

- la trace est structurée par tous les blocs de reprises, alors que l'écran virtuel n'est structuré que par les blocs du dialogue (en excluant les blocs de reprise créés à la compilation)
- à l'exécution d'un fin de bloc non rémanent, toutes les structures internes sont effacées de l'écran virtuel, alors que les données correspondantes restent dans la trace
- la trace ne contient pas les valeurs en édition seule
- l'écran virtuel permet d'accéder aux structures associées aux valeurs, alors que la trace ne contient que ces valeurs.

Ces différences reflètent l'utilisation particulière de chacune des structures :

- l'écran virtuel permet de gérer l'écran réel sans exiger une taille mémoire excessive
- la trace permet d'éviter des saisies inutiles lors des reprises.

Par contre, les deux structures contiennent les numéros d'ordre des données échangées en saisie ou édition-saisie. Elles vont en effet coopérer au moment du déclenchement d'une reprise : la donnée modifiée, repérée sur l'écran virtuel, va devoir être localisée dans la trace du dialogue.

La localisation dans l'écran virtuel d'une donnée modifiée par l'utilisateur sur l'écran réel pose le problème de l'association écran virtuel/écran réel.

4.2.3 - *Association écran virtuel/écran réel* :

Pour chaque rubrique de saisie, le programmeur a défini une fenêtre d'affichage. Lors de l'exécution du programme, seules les dernières fenêtres sont réellement affichées. Lorsqu'une nouvelle fenêtre est demandée, s'il n'y a plus de place sur l'écran, on efface la fenêtre la plus ancienne.

Ceci se généralise aux sous-grilles répétitives : la fenêtre associée à la sous-grille représentera l'écran vis-à-vis des fenêtre internes (associées aux rubriques).

Pour des raisons évidentes de clarté d'affichage, les fenêtres seront toujours suffisantes pour éviter de "couper" une rubrique élémentaire.

Il semble plus simple que toute fenêtre soit un pavé rectangulaire de largeur égale à la largeur de l'écran. L'écran réel se déplace donc dans les seuls sens "haut" et "bas" par rapport à l'écran virtuel. La définition de fenêtre juxtaposées ou de fenêtres plus larges que l'écran risque d'être déroutante pour l'utilisateur, puisqu'il lui faudra gérer un déplacement supplémentaire "gauche" et "droit". Par contre, ceci rendrait la définition des cadres indépendante de la largeur de l'écran.

L'affichage d'un cadre d'écran consistera à :
- créer éventuellement la place nécessaire par effacement de cadres plus anciens
- écrire le cadre dans la mémoire d'écran.

Cet affichage aura lieu à l'exécution d'une saisie ou édition, mais également lors de la remontée dans l'écran virtuel (utilisation de la touche "tabulation arrière" par l'utilisateur).

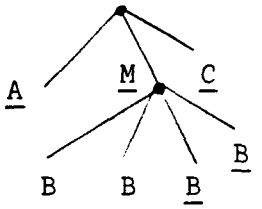
A tout moment, l'écran virtuel devra comporter les renseignements permettant de retrouver ce qui est réellement affiché.

Les feuilles de l'écran virtuel sont des rubriques simples. Une sous-grille sera donc représentée par un sous-arbre. Chaque feuille comportera un booléen indiquant si la rubrique est ou non réellement affichée.

Si la fenêtre associée à une sous grille est affichée mais n'est pas suffisante pour contenir toute la sous-grille, certaines rubriques seront réellement affichées alors que d'autres ne le seront que virtuellement.

exemple (les rubriques soulignées sont réellement affichées)

saisie (A)
saisie (M : [ ] de B)
(fenêtre de 2 rubriques)
saisie (C)



L'écran virtuel doit permettre de retrouver une fenêtre associée à un noeud, et le booléen correspondant. C'est le seul cas où des données sont attachées à des noeuds. Ceci n'est utile que pour les sous-grilles répétitives. D'autre part, il faudra gérer le déplacement de la fenêtre à l'intérieur de la sous-grille, c'est à dire, du point de vue de l'utilisateur, le défilement en rouleau à l'intérieur de la fenêtre associée. Ceci pourrait se trouver à plusieurs niveaux, mais la presque totalité des cas se fera à un ou deux niveaux (fenêtres de rubriques incluses dans une fenêtre de sous-grille).

L'association de fenêtres aux blocs de programme généraliserait ce mécanisme à un nombre de niveaux quelconque.

L'écran virtuel sera donc une arborescence contenant les données virtuellement affichées et les "renseignements d'affichage" associés. Sa structure reflète la structure du programme de dialogue, à l'exception des blocs de reprise ajoutés à la compilation.

En tenant compte d'un parcours très particulier des deux arborescences (écran virtuel et trace), une implémentation économique peut être proposée.

4.3 - Implémentation minimale et extensions possibles

4.3.1 - *Implémentation minimale*

Le dialogue est supposé résider sur le processeur de traitement, le termina se chargeant de l'exécution des primitives de saisie ou d'édition. La trace du dialogue sera donc sur le processeur de traitement, tandis que le fichier des rubriques de saisie ainsi que l'écran virtuel seront gérés par le terminal.

Le logiciel proposé (en pseudo-langage) a pour but de montrer la simplicité des mécanismes pour une implémentation minimale. On ne permet pas au programmeur d'associer de fenêtres aux sous-grilles en table. Celles-ci s'afficheront donc "in extenso" et il n'y aura pas de défilement "en rouleau" à l'intérieur d'une fenêtre. Le seul niveau de défilement en rouleau est celui de l'écran virtuel par rapport à l'écran réel. La définition d'une sous-grille en table aura donc le même effet que la définition explicite de l'itération (saisie (A[1..5] de B) a le même effet que pour i = 1 à 5 début saisie (B) fin rémanent).

4.3.1.1 - Représentation linéaire des arborescences

L'intérêt de cette implémentation est de détailler les mécanismes proposés, et en particulier de montrer que la trace et l'écran virtuel sont parcourus "séquentiellement" : on n'accède qu'aux feuilles et d'une feuille à l'autre de proche en proche. Les deux "arborescences" seront donc en fait implémentées par des listes. L'utilisation de piles permettra de tenir compte de l'imbrication des blocs.

```
Exemple : ... début 1 saisie (A, B)
(blocs 1 et 2 non          début 2 saisie (C)
 rémanents)                fin 2
                               saisie (D)
                               ...
           fin 1
```

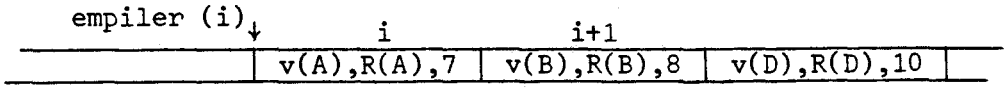
trace correspondante (la structuration par les blocs de reprise se fait à l'aide d'une pile) v(x) = valeur de x.

	v(A)	v(B)	v(C)	v(D)					
numéro d'ordre :	7	8	9	10	11	12	13	14	

Lorsqu'une variable n'est pas d'un type simple (enregistrement...), la trace contient non pas sa valeur mais l'accès à sa valeur.

Ecran virtuel (la structuration par les blocs non rémanents se fera par empilement de l'indice dans la liste du début de chacun de ces blocs).

En plus de la valeur des variables (v(X)) l'écran virtuel doit contenir l'accès à la rubrique associée (R(X)) ainsi que leur numéro d'ordre dans la trace (O(X))



A la rencontre d'une fin de bloc non rémanent, on efface la fin de l'écran virtuel à partir du dernier début (dépiler).

Pour la gestion de l'affichage, les variables suivantes sont nécessaires :

- espace libre : l'espace libre sur l'écran, compté en nombre de lignes
- rubrique haute : indice dans l'écran virtuel de la rubrique la plus "haute" sur l'écran (la plus ancienne)
- rubrique basse : même chose pour la rubrique basse. Ceci sera utile en cas de remontée dans l'écran virtuel
- rubrique courante : indice de la rubrique actuellement pointée par le curseur

4.3.1.2 - Procédures d'échange

Deux familles de procédures sont à étudier :

- les procédures de construction des structures
- les procédures de relecture et de reprise

Procédure saisie (A, ordre (A))

A est une rubrique, c'est à dire une variable d'un type simple (entier...) ou composé (enregistrement) et ses attributs d'affichage.

Le numéro d'ordre est l'indice où sera rangé cette valeur dans la trace.

A la rencontre de cette primitive par la procédure de dialogue, il faut :

- déclencher l'affichage par le terminal du cadre de rubrique
- effectuer la saisie effective par le terminal (contrôle syntaxique)
- ajouter la saisie à l'écran virtuel
- transmettre la saisie au dialogue et l'ajouter à la trace

Procédure afficher-cadre (rubrique de saisie)

Il faut (au besoin) créer l'espace suffisant à la fenêtre associée puis écrire le cadre dans la mémoire d'écran. La création d'espace se fait par effacement de la rubrique la plus extrême sur l'écran. En cas de construction de l'écran (saisie), la rubrique extrême est la rubrique haute.

```
afficher-cadre (nom de rubrique)
début tant que espace - libre < fenêtre (rubrique)
      faire décaler ligne à ligne la mémoire d'écran en écrasant la rubrique haute ;
            espace libre := espace libre + fenêtre (rubrique haute)
            rubrique haute + := 1
      fait ;
  écrire le cadre dans la mémoire d'écran après rubrique basse :
  rubrique basse + /+ 1 ; estpace libre := espace libre - fenêtre (rubrique)
fin
```

La saisie avec contrôle syntaxique à la volée suppose l'existence d'un analyseur syntaxique "temps réel", disposant des messages d'erreurs associés aux types. Ces messages seront (au besoin) affichés dans une zone réservée de l'écran (ligne système...).

Cette donnée est ajoutée à l'écran virtuel ainsi que le nom de rubrique associé et son numéro d'ordre dans la trace.

La donnée est ensuite transmise au dialogue.

Les procédures d'édition sont beaucoup plus simples. Les procédures d'édition-saisie sont très similaires.

L'utilisateur peut "monter" ou "descendre" dans l'écran virtuel grâce aux touches TABULATION AVANT ou ARRIERE. Lorsqu'il sort d'une rubrique, les procédures Remontée ou Suite seront appelées.

```
Remontée : début rubrique   courante - := 1
                si rubrique courante < rubrique haute
                   alors afficher (rubrique courante) ; rubrique haute -:=1 fsi
           fin
```

La procédure afficher (rubrique)

consiste à afficher le cadre et le remplir grâce aux données de l'écran virtuel. Suivant le sens de parcours de l'écran virtuel (remontée ou suite), la rubrique à effacer en cas de manque de place sera la rubrique haute ou basse.

Lorsque l'utilisateur modifie une donnée, une reprise est déclenchée. Le terminal efface la fin de l'écran virtuel et transmet au dialogue la donnée modifiée et son numéro d'ordre.

Le dialogue déterminera alors le lieu de reprise (nombre de blocs de reprise dont il faut sortir).

Reprise (nouvelle valeur, numéro) : numéro est le numéro d'ordre de la donnée modifiée. Il faut :

- Déterminer le nombre de blocs dont il faut sortir
- Détruire la fin de la trace
- Modifier la valeur dans la trace
- Exécuter les traitements de reprise des blocs.

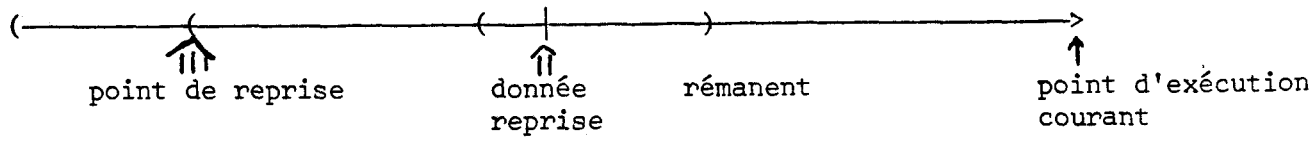
Le lieu de reprise est le début de bloc contenant simultanément le point d'exécution courant (queue de trace) et la donnée modifiée. A chaque rencontre de début de bloc de reprise, on a empilé le numéro d'ordre courant. On dépile à chaque rencontre de fin de bloc de reprise.

Pour trouver le lieu de reprise, il suffira de dépiler jusqu'au premier début inférieur au numéro d'ordre de la donnée modifiée. Le nombre de dépilement est le nombre de blocs de reprise dont il faut sortir (en exécutant les traitements de reprise associés).

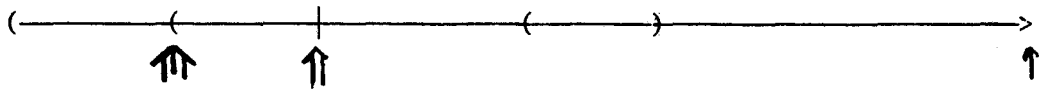
```
Reprise (nouvelle valeur, numéro)
début
        contenu (numéro) ← nouvelle valeur ;
        détruire la trace après numéro ;
        compteur ← 0 ;
        faire reprise ← dépiler ; compteur + := 1 jusqu'à  reprise ≤ numéro
        déclencher l'exécution des traitements de reprises des v(compteur) blocs
        englobants.
        indice courant ← reprise
fin
```

Ce traitement, systématique, n'a pas à tenir compte des différents cas possibles :

- la donnée modifiée est interne à un bloc déjà terminé (bloc rémanent)



- le bloc englobant immédiatement la donnée n'est pas encore terminé



- certains blocs compris entre la donnée modifiée et le point de reprise ont du être franchis (non création de bloc de reprise par coût de mémorisation excessif).

L'exécution des traitements de reprise associés pourra se faire par déclenchement d'un mécanisme de type exception ADA. La propagation d'une reprise d'un bloc à un bloc englobant devra se faire avec passage d'un paramètre : nombre de blocs restant à reprendre. Lorsque le point de reprise sera atteint, le contexte initial de ce bloc aura été restitué.

Il faut alors se replacer sur la donnée modifiée par l'exécution des instructions intermédiaires.

Pour que ceci soit transparent à l'utilisateur, il faut qu'avant chaque édition ou saisie effective, le programme vérifie que l'on se trouve bien en queue de trace. Sinon, la saisie se fait par lecture dans la trace et l'édition ne se fait pas.

4.3.2 - Extensions

Cette pseudo-implémentation minimale montre la simplicité des mécanismes dans un cas très restrictif : il n'y a jamais défilement "en rouleau" à l'intérieur d'une fenêtre, ce qui interdit, par exemple, l'association d'une fenêtre aux grilles dynamiques. Ceci est pourtant prévu dans les spécifications proposées au 3ème chapitre.

4.3.2.1 - Extensions respectant les spécifications

Il est pourtant bien agréable de pouvoir diviser l'écran en "sous-écrans". Dans certains cas (bien rares, sans doute), ceci peut se généraliser à plusieurs niveaux.

Pour un niveau donné (une fenêtre sous-écran ou l'écran) les procédures de gestion de l'écran virtuel restent valables. Le passage d'un niveau à un autre exige.

- la gestion des variables en pile (espace libre, rubrique haute, rubrique basse)
- la gestion d'un écran virtuel arborescent.

En effet, une fenêtre sous-écran sera représentée par un sous-arbre. A la racine de ce sous-arbre sera attachée la dimension de la fenêtre. Les variables rubrique-haute et rubrique-basse pourront pointer vers ces sous-arbres.

Bien que la gestion en sera plus complexe, un tel écran virtuel reste implémentable sur un terminal évolué. Ceci permettrait également d'associer des fenêtres aux différents blocs de programme ou procédures.

De même, si le terminal peut le supporter, il est préférable que tout ou partie des procédures de dialogue y soient résidentes. Ceci diminuera d'autant la charge du processeur central et des transmissions.

Seul le programme de traitement sera exécuté par le processeur de traitement (ou le réseau ...). Dans certains cas, il n'y aura d'ailleurs pas de traitement (transactions sur des fichiers du terminal ...)

Les spécifications du langage proposé précisent que la définition des fenêtr associées aux rubriques ou aux tables est à la charge du programmeur. On pourrait également donner à l'utilisateur averti la possibilité de modifier cette "mise en écran".

4.3.2.2 - Mise_en écran dynamique

Les fenêtres associées aux feuilles (rubriques simples) ou aux noeuds (sous-grilles ou blocs de programme) de l'écran virtuel constituent une transformation permettant de "projeter" l'écran virtuel sur l'écran réel. Cette transformation ne peut pas être optimale: il faut tenir compte à la fois des contraintes de taille d'écran et des besoins de l'utilisateur. Hors, ces besoins peuvent évoluer au cours d'une exécution du programme : à un moment donné, l'utilisateur préférera une vue synthétique, les fenêtres associées aux tables étant réduites à une seule rubrique, ensuite il désirera visualiser une table "in extenso" ...

Les spécifications du langage précisent que cette transformation est définie statiquement à l'écriture de l'application, grâce à un éditeur d'écran interactif. Il n'est en effet pas raisonnable d'imposer à l'utilisateur un

"langage" supplémentaire permettant de définir et modifier la "projection". L'utilisateur a quand même une influence sur le domaine de définition de la transformation par le biais des retours dans l'écran virtuel. Par contre, ce langage peut être mis à la disposition d'utilisateurs plus avertis. En particulier, ces utilisateurs seront familiarisés avec la notion d'arborescence. Il y aura alors deux modes possibles au cours d'une exécution du programme d'application. Le mode normal, c'est-à-dire saisie avec visualisation classique (les retours dans l'écran virtuel étant tout de même permis), et un mode commande, permettant de modifier la transformation. En mode commande, l'utilisateur n'effectue plus de saisie mais accède aux fenêtres associées aux noeuds de l'écran virtuel. Dans la version précédente, il n'y a qu'une version de fenêtre par sous grille (ou bloc ...). Si plusieurs noeuds correspondent à cette sous-grille, on peut imaginer de dupliquer la fenêtre à chaque noeud, ce qui permettrait des modifications différentes sur chaque version. Ces modifications seraient provisoires, une nouvelle version trouvant la fenêtre originelle. La possibilité de reporter une modification sur la fenêtre attachée à la sous-grille (ou bloc) permettrait de redéfinir d'une manière permanente la "projection" d'une application. Le langage de commande sera un véritable éditeur de texte interactif manipulant une arborescence. Une réalisation proche, mais en contexte graphique, a été étudié par PARENT dans [PAREN 82]. Une synthèse des deux approches pourrait peut-être résoudre les problèmes des applications mixtes (dialogues alphanumériques et graphiques). Les idées sous-jacentes sont identiques : l'utilisateur crée une arborescence (saisie de caractères ou d'images) à laquelle il applique une transformation (projection dans une fenêtre ou rotation ...).

La modification par l'utilisateur averti des fenêtres attachées aux noeuds de l'écran virtuel est donc envisageable.

Mais l'ensemble des mécanismes proposés présente encore un grave inconvénient : en cas de reprise toutes les saisies postérieures à la donnée modifiée sont perdues, ce qui risque de dissuader l'utilisateur de corriger des erreurs qu'il jugera mineures. Le problème est de déterminer, parmi ces données, celles qui peuvent encore conserver un sens. Par exemple, en cas de changement de données de définition, les données de traitement associées sont à éliminer. Une manière d'éliminer les données obligatoirement invalides est de gérer une trace d'exécution plus complète.

4.3.2.3 - Utilisation de toutes les données déjà saisies

L'adjonction à la trace de "données de contrôle" va permettre de réutiliser parmi les données saisies postérieurement à la donnée modifiée, celles qui pourraient encore avoir un sens.

Parmi les variables du programme, quelles sont celles qui peuvent influencer la réutilisation des saisies ? L'exemple évident est celui du type de transaction demandé. D'autres ont un effet moins immédiat.

En cas de reprise, la "structure" de l'exécution précédente peut ne pas se modifier ; c'est sans doute le cas le plus courant ; toutes les données saisies sont donc valides, et il ne faut en supprimer aucune. Mais cette structure peut changer : des blocs conditionnels ne seront pas ré-exécutés, des blocs itératifs seront exécutés plus ou moins souvent ...

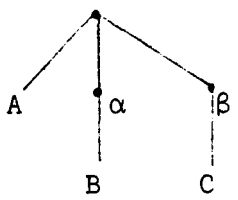
Il suffit donc de retenir la valeur des variables de contrôle commandant l'exécution des blocs : booléens, dimensions ... A la réexécution des instructions comprises entre la saisie de la donnée modifiée et le point correspondant à la fin de la trace, il faudra comparer les nouvelles valeurs aux anciennes.

Les sous-arbres correspondant aux blocs non réexécutés seront détruit Les saisies susceptibles d'être encore valides seront affichées avec demande de confirmation.

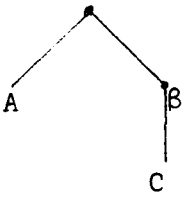
exemple :

```
début saisie (A) ;
      α = f(A)
   si α alors saisie (B) fsi ;
   si β alors saisie (C) fsi
fsi
```

trace initiale



trace possible après reprise sur



La valeur C n'a pas été ressaisie.

A l'inverse, il faut pouvoir ajouter des sous-arbres dans la trace (blocs non exécutés initialement). La gestion de la structure devient donc assez complexe.

La réutilisation partielle de blocs répétitifs ou de sous-grilles en table pose le problème de ce qu'il faut conserver ou non.

exemple : saisie (A(1..n))
En cas de diminution du nombre d'itération au cours d'une reprise, il faudra choisir les valeurs à éliminer. On peut évidemment prendre la convention de ne garder que le début, ou bien laisser le choix à l'utilisateur, solution plus souple mais plus longue à mettre en oeuvre. De même en cas d'augmentation de la dimension d'un tableau, les valeurs ajoutées pourront être ajoutées en queue ou au contraire insérées.

L'écran virtuel pourra être, comme la trace, profondément modifié, par la suppression ou l'adjonction des sous-arbres en des points quelconques de la structure.

On peut laisser à l'utilisateur la possibilité de provoquer une reprise lorsque l'exécution est déjà en mode reprise. Il faudra alors gérer des sous-arbres non complets.

La réutilisation des saisies qui suivent la saisie modifiée est donc possible. La gestion est évidemment plus complexe, mais surtout elle ne peut plus être systématique. L'utilisateur doit pouvoir confirmer ou non ces valeurs, et dans certains cas (changement de dimension de tableau ...), ceci nécessite un véritable dialogue "parasite". Cette dernière possibilité n'est à mettre en oeuvre que dans des cas bien spécifiques. Par contre, ce mécanisme est tout à fait justifié lorsque la modification n'a pas entrainé de changement de structure : et ceci concerne sans doute la majorité des cas, puisque le nombre de données "neutres" vis à vis du déroulement du programme est en général bien supérieur à celui des données "à risque".

4.3.3 - *ADA et le langage de dialogue*

Bien que les concepteurs de compilateur Ada semblent rencontrer quelques problèmes, ce langage est unanimement reconnu comme le langage des années 80. De nombreux auteurs se sont attachés à montrer la puissance des concepts qu'il intègre. Une étude critique en a été publiée : "le langage Ada : manuel d'évaluation " [VERRA 82]

Pour mémoire, on peut citer :

- la modularité
- la généricité
- le mécanisme de rendez-vous
- le mécanisme d'exception.

Malgré certaines critiques (plus ou moins synthétisées dans le manuel d'évaluation) les recherches dans le domaine langages/système tendent donc à se situer par rapport à cette référence. Ce travail ne fera pas exception à la règle.

L'intégration du langage proposé dans un environnement Ada peut se traiter suivant deux aspects :

α) l'utilisation des outils Ada par le langage de dialogue ; un mécanisme du type "exception" s'adapte bien aux traitements de reprise

β) l'utilisation de procédures de dialogue par des programmes Ada ; trois points seront évoqués : les limites imposées aux paramètres, la génération des spécifications des procédures et l'utilisation des "rendez-vous" pour permettre un parallélisme dialogue/traitement.

α) Les exceptions ADA fournissent un mécanisme d'écriture de logiciels resistants aux erreurs. Elles sont déclenchées soit explicitement (raise) soit implicitement (exceptions prédéfinies, telles que division par zéro ...). Lorsqu'une exception est déclenchée dans une unité, elle est propagée jusqu'au récupérateur qui contient le traitement qui lui est associé. Après récupération, le contrôle est rendu à l'unité englobante.

Les exceptions ne peuvent avoir de paramètres. Du point de vue compilation, elles sont considérées comme des constantes de type énuméré prédéfini. Mais le traitement d'exception a les mêmes droits (visibilité ...) que les instructions de l'unité. On peut donc simuler les paramètres par des variables globales.

Enfin l'association entre une exception et son récupérateur est dynamique, puisqu'elle peut dépendre d'un ordre d'appel des unités considérées, et pas seulement du lieu des déclarations.

Vis-à-vis du langage de dialogue, les exceptions peuvent être utilisées pour l'implémentation des reprises.

Bien qu'elles soient très proches l'une de l'autre, on pourra distinguer les reprises dues au non respect d'une spécification de cohérence des reprises provoquées par l'utilisateur lui-même. Il faut d'ailleurs noter que dans la version initiale du langage ADA ("green"), les assertions de cohérence étaient prévues.

Le traitement de reprise lié à une violation d'une règle de cohérence devra d'abord déterminer la donnée erronée ; si la règle spécifie un "ensemble de cohérence", l'utilisateur devra choisir parmi cet ensemble la donnée qu'il voudra modifier. Un artifice visuel pourra l'aider dans cette tâche (clignotement des données suspectes ...). Ce procédé sera suffisamment systématique pour ne pas nécessiter du programmeur plus que la spécification de l'assertion de cohérence. Attaché à chaque assertion, ce traitement peut ne pas faire partie du mécanisme de reprise au sens récupérateur d'exception.

Une fois la donnée modifiée (que ce soit spontanément ou par obligation), la reprise proprement dite peut être déclenchée.

Le numéro d'ordre associé à la donnée dans la trace et l'écran virtuel permet de déduire le nombre de blocs de reprise emboités dont il faut sortir (chapitre implémentation). Ce nombre sera une variable globale accessible par tous les blocs de reprise.

Le déclenchement de la reprise se fera grâce à une instruction raise REPRISE. Chaque bloc de reprise possédera un récupérateur de cette exception. Le traitement associé consistera à restituer le contexte initial du bloc. En fin de traitement, le nombre de blocs à reprendre sera décrémenté. S'il n'est pas nul, l'exception sera propagée au bloc immédiatement englobant par l'instruction raise sans identificateur d'exception. Lorsqu'il devient nul, le contrôle sera redonné au début du bloc par une rupture de séquence ("goto" DEBUTBLOC). Il faut noter que la restitution du contrôle à l'intérieur du bloc est interdite : l'étiquette DEBUT BLOC sera placée immédiatement devant le début.

Le positionnement à l'endroit de reprise exact (qui n'est pas toujours le début de bloc) se fera sans intervention de l'utilisateur grâce à la

relecture dans la trace des 'saisies intermédiaires'.

exemple d'implémentation ADA des blocs de reprise :

```
REPRISE 1
begin
          REPRISE 2
          begin
          exception : when REPRISE => ... ;
          end
exception : when REPRISE => Restitution de contexte ; NBREBR := NBREBR - 1 ;
                    if NBREBR = 0 then goto  REPRISE 1 else raise endif
end
```

La restitution du contexte présente deux aspects : restitution du contexte interne (par exemple par dépilement si le contexte est géré en pile) et restitution du contexte externe ("opérations inverses")

β) Utilisation de procédures de dialogue par les programmes ADA

Les entrées-sorties du langage ADA ont été très controversées et plusieurs fois remaniées. On leur reproche en particulier leur trop grande rusticité, qui subsiste d'ailleurs dans les versions définitives. Ce choix est par contre justifié par les concepteurs en invoquant la possibilité de construire des paquetages d'entrées-sorties plus évoluées à partir des outils rudimentaires qui sont proposés. Ce qui posera évidemment des problèmes de compatibilité.

Dans une conclusion très critique, le manuel d'évaluation propose une alternative par le choix d'un autre modèle d'entrée-sortie (chapitre rédigé par Lecarme-Rousseau). La notion d'entrée-sortie se scinde en trois domaines de communication ; un programme ADA peut mémoriser des données (fonctions d'un Système de Gestion de Bases de Données), peut échanger des données avec des sites éloignés (fonctions d'un réseau informatique) et enfin acquérir ou restituer des valeurs dans l'environnement des utilisateurs.

Ce dernier système d'acquisition-restitution doit permettre de gérer un dialogue "ergonomique" : contrôles de syntaxe, de sémantique, facilités de présentation des documents ...

Le langage proposé par cette thèse répond en grande partie à ces objectifs, tout au moins pour ce qui concerne les échanges alphanumériques. Il est donc intéressant d'étudier les possibilités d'utilisation de procédures de dialogue

par les programmes ADA. Cette étude sera néanmoins très succincte : les techniques d'appel de procédures non Ada dépendent plus du compilateur ADA que du langage appelé.

La possibilité de tels appels est prévue en ADA par le pragma INTERFACE. La complexité de l'interface entre le langage de dialogue et ADA sera fonction d'un certain "degré de similitude" entre les deux langages. Celui-ci peut s'étudier suivant deux aspects : les possibilités de paramétrage et la possibilité de générer les spécifications (au sens Ada) des procédures de dialogue.

Pour laisser une certaine liberté aux procédures de dialogue, il faudra disposer de types de paramètres très "souples" : tableaux non contraints ... Sur ce point, le langage Ada n'impose pas de restrictions : les paramètres formels peuvent être d'un type quelconque du langage. L'impossiblité de passer une procédure en paramètre ne présente aucun inconvénient pour le sujet qui nous intéresse.

La possibilité d'associer une spécification Ada à une procédure de dialogue faciliterait les contrôles statiques, et donnerait une certaine "orthogonalité" à l'association de ces deux langages. Un générateur de spécifications ne devrait pas être difficile à écrire : le langage du dialogue étant très spécialisé, les constructions qu'il permet sont assez limitées. La seule démonstration de cette opinion serait évidemment l'écriture du générateur, ce qui suppose une définition préalable du langage plus rigoureuse (grammaire ...). La possibilité inverse de vérifier l'adéquation d'une procédure de dialogue à une spécification écrite par un programmeur Ada n'est pas à négliger.

Une dernière possibilité à offrir aux programmes Ada serait la disjonction entre l'instruction de déclenchement d'une procédure de dialogue et l'instruction de prise en compte des données qu'elle délivrera. On pourrait donc appeler une procédure de dialogue dès que possible et n'attendre le retour qu'en cas de nécessité. Cette particularité permettrait un parallélisme traitement/dialogue qui se justifie pour plusieurs raisons : le dialogue est une phase très lente du déroulement d'un programme, il devrait être pris en charge par un matériel adapté (terminal intelligent) différent du système de traitement ... Le mécanisme ADA des Rendez-vous permet sans difficulté l'écriture d'un interface gérant la synchronisation et la communication des deux processus dialogue et traitement.

On peut donc dire que le langage de dialogue s'intégrerait aisément dans un environnement de programmation Ada. Ceci correspondrait tout à fait au souhait énoncé par ROUSSEAU dans le manuel d'évaluation : celui d'une organisation à plusieurs niveaux permettant une spécialisation des programmeurs et facilitant la transportabilité.

4.4 - Conclusion

Le respect des spécifications du langage défini au chapitre 3 nécessite donc la gestion de deux structures de données : la trace du dialogue et l'écran virtuel. Ce sont des arborescences reflétant la structure du programme. Cette structure est soit initiale, soit surajoutée à la compilation pour favoriser l'utilisateur en cas de reprise. Une implémentation minimale peut se faire en liste, l'imbrication des blocs étant alors gérée à l'aide d'une pile. La possibilité d'attacher des valeurs aux noeuds qui ne sont pas des feuilles interdit d'utiliser l'implémentation minimale mais offre trois axes de développement intéressants :

- association de fenêtres aux tables et aux blocs de programme
- mise en écran modifiable par l'utilisateur grâce à un langage de commande
- ré-utilisation des données qui suivent une donnée modifié lors d'une reprise.

Enfin, l'intégration du langage dans un environnement de programmation Ada est tout à fait envisageable, et résoudrait certaines faiblesses des entrées-sorties Ada.

5 - VERS UN MODÈLE GÉNÉRAL DE L'INTERACTION

Le problème de l'interaction homme/machine peut s'énoncer en termes de compromis : d'un côté, la pensée humaine, intuitive, "associative", non formalisée ... de l'autre, un système informatique, aux algorithmes définis d'une manière relativement stricte et figée. L'utilisateur acceptant de moins en moins (et à juste titre) de se plier aux exigences du système, il faut proposer un modèle d'interface à mi-chemin entre ces deux mondes. Les réalisations actuelles sont en général très dépendantes d'un matériel ou d'un langage particulier, et règlent donc le problème au "coup par coup". Peut-on proposer un modèle général de l'interaction ?

Le modèle d'organisation graphique (au sens "structure" et non pas "image") semble assez séduisant. Introduit dans les années 70 par CODD pour les bases de données (modèle relationnel), il est repris actuellement pour formaliser la structure des dialogues (GESCRAN ...). Hélas, la richesse des possibilités de description offerte par le graphe se paie à la programmation, par une complexité qui dépasse très vite les capacités humaines (la matrice de transition des dialogues en GESCRAN risque d'atteindre des proportions parfois excessives). Même dans les cas de figure raisonnables, un graphe est toujours difficile à appréhender. On peut s'attacher à comprendre un état, un certain nombre de transitions, mais dominer l'ensemble de la structure est rarement possible.

Le modèle arborescent représente, à mon avis, un compromis plus raisonnable. Plus limitatif que le graphe (dont il n'est qu'un cas très particulier), il présente néanmoins une grande souplesse de description des dialogues. Ce travail a d'ailleurs montré comment la gestion d'une trace arborescente et de blocs de reprises imbriqués peuvent simuler les transitions vers un endroit quelconque du dialogue lors des reprise. Mais surtout, le modèle hiérarchique qui sous-tend toute arborescence est parfaitement adapté aux types de raisonnement analytique ou synthétique : on peut s'occuper d'un niveau donné, dans le détail, c'est à dire d'un noeud et de ses descendants sans tenir compte des ascendants et des frères. Ou au contraire garder une vue globale sans se soucier du détail de chaque sous arbre. Cette démarche tend à être unanimement acceptée pour l'analyse et la programmation (dites "structurées").

Mais le modèle hiérarchique (arborescent), représenté par chaque programme structuré est encore bien contraignant pour l'utilisateur. A défaut de lui proposer la machine idéale que permettra peut-être un jour les progrès de l'intelligence artificielle, on peut tenter d'assouplir le dialogue défini par l'arborescence figée qu'est un programme compilé.

Un modèle d'interface arborescent pourrait présenter deux aspects, correspondant à la dichotomie compilation/exécution : une arborescence figée, produit de la compilation et qui sera le "squelette" de chaque exécution, et une arborescence dynamique qui rendra compte des événements d'une exécution particulière. La souplesse d'un dialogue dépendra alors, d'une part des outils donnés au programmeur pour "enrichir" l'arbre statique, et d'autre part des possibilités laissées à l'utilisateur d'agir sur l'arbre dynamique. L'utilisateur n'ayant pas, a priori, à connaitre la notion d'arborescence, ces actions seront bien sûr implicites.

Tout ceci n'est pas nouveau et, par exemple, la notion d'arbre syntaxique a déjà été utilisée pour faciliter les différentes phases de la compilation (décoration de l'arbre par des attributs sémantiques ...). Un certain nombre de travaux effectués à l'Université de Lille ont préconisé l'utilisation explicite d'arbres pour résoudre des problèmes liés aux entrées-sorties : un modèle d'entrée-sortie pour une machine parallèle (SAUGE) [LECOU 82], un éditeur graphique interactif [PAREN 82] ...

Une thèse [FRANK 82] a également proposé des mécanismes pour assouplir la structuration des programmes transactionnels en contexte réparti. Ceci donne une plus grande liberté de définition de l'"arbre statique". Il est donc intéressant de confronter ces trois axes de recherches au modèle arbre statique/arbre dynamique évoqué.

5.1 - Modèle d'entrée-sortie pour une machine parallèle : SAUGE

Les entrées-sorties sont essentiellement séquentielles : leur traitement par une machine parallèle risque donc de causer une perte de performance inacceptable. Un modèle d'entrée-sortie pour SAUGE (machine parallèle utilisant l'assignation unique) a été proposé dans [LECOU 82]. Ce modèle, très général, est applicable à d'autres types de parallèlisme. Les entrées-sorties conversationnelles au cours d'une exécution parallèle n'ayant pas de sens, toutes les

saisies sont supposées acquises au préalable, et toutes les éditions se font en bloc après l'exécution. Le modèle proposé ne s'intéresse donc pas à la communication homme-machine mais à la coopération entre trois processus : entrée, sortie, exécution. Le processus exécution consomme les valeurs en entrée et produit les valeurs en sortie. Le problème à résoudre sera l'identification des valeurs consommées et l'ordonnancement des valeurs produites. Des identificateurs devront donc être générés en cours d'exécution, à un coût raisonnable. La solution proposée est basée sur des arborescences pour stocker les valeurs à lire et à écrire : les processus d'entrée et de sortie étant séparés et symétriques, il y aura un arbre d'entrée et un arbre de sortie. En résumé, le mécanisme de sortie (et respectivment d'entrée) fait intervenir un arbre statique, l'"arbre programmatique" qui indique l'imbrication des séquences, des boucles et des alternatives, et un arbre dynamique, résultat d'une exécution. Chaque arbre dynamique est un sous-arbre de l'arbre infini (virtuel) représentant toutes les exécutions possibles (nombres infini d'itérations ...). L'arbre statique est le squelette de cet arbre virtuel : il contient le premier des noeuds correspondant aux itérations ...

L'arbre dynamique contiendra, en plus des données de sortie, les données de contrôle : valeur du booléen commandant une alternative ... Ce mécanisme a été réutilisé pour résoudre, dans le langage de dialogue, le problème de réutilisation des données postérieures à une donnée modifiée lors d'une reprise (trace améliorée).

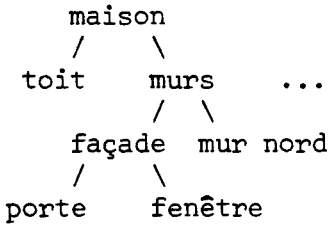
Ce modèle permet donc une interaction de type parallèle entre un processus de traitement et des processus d'entrée et de sortie. Cette interaction ne faisant pas intervenir l'utilisateur, les problèmes ne relèvent pas d'une "ergonomie du dialogue". Mais le modèle proposé montre que la définition d'un arbre statique et d'un arbre dynamique permet une souplesse extrême dans le déroulement des entrées-sorties.

Une autre utilisation des arborescences a été proposée dans le domaine graphique ([PAREN 82]).

5.2 - Editeur d'images interactif

Cette étude traite d'un éditeur graphique sur STYX (système de conception d'images par taches de couleur). L'éditeur permet la synthèse d'image en interactif, et laisse une très grande liberté de manoeuvre à l'utilisateur qui

devra être familiarisé avec la notion d'arborescence. Au niveau des images, celle-ci est assez intuitive : une maison, c'est un toit, quatre murs ...



Chaque image est une arborescence cataloguée. L'utilisateur peut définir la structure d'un objet et sa mise en écran par l'adjonction d'attributs graphiques à cette structure (forme, couleur, texture ...). Il travaille sur une image (l'écran) qui est différente de l'objet image (structure décorée). Le problème est donc d'associer à chacune des actions sur l'écran une action sur la structure. De nombreuses possibilités sont laissées à l'utilisateur :
- il dispose de différents types d'objets : les objets images, les objets motifs (briques ...), les objets récursifs (deux miroirs se réflétant à l'infini), et les objets patrons (incomplétement spécifiés : des "modèles").
- il dispose de nombreuses commandes : des commandes de parcours de la structure de l'objet, qui permettent de changer le contexte de visualisation (porte seule ou porte avec "son" mur ...) ou l'objet sur lequel on travaille (la maison en entier ou le toit ...) ; des commandes de modification de la structure (rattachement de noeuds, clivage, déplacement, insertion, suppression ...), des commandes de modification des attributs graphiques associés aux noeuds (transformations géométriques, fenêtre associée, couleur ...).

L'ensemble de ces commandes permet de construire une image à partir d'éléments simples ou en décomposant l'objet complet initial.

Cet éditeur propose donc une véritable méthodologie de construction d'images considérées en tant qu'arborescences. Le langage associé a été formalisé à l'aide d'un outil théorique, les transducteurs d'arbres ([DAUCH 77], Engelfriet. de manière à fournir une référence rigoureuse pour l'implémentation et l'usager. Le langage de commande peut paraître complexe au non-initié, mais ceci n'est que le revers de la puissance des concepts qu'il met en oeuvre : récursivité, arborescence ...

Cette application diffère quelque peu du modèle général de l'interaction qui est proposé (arbre statique / arbre dynamique). L'arbre statique n'existe pas a priori, puisque l'utilisateur dispose des commandes primitives : il n'y a pas de programme prédéfini (l'enregistrement d'une suite de commande pourrait permettre de créer des dessins animés).

Une session correspond à une création et/ou une modification d'une image, ce qui se traduit par une manipulation d'arbre. L'utilisateur manipule donc des images, cataloguées dans une base de données image.

Dans un tout autre contexte (bases de données réparties), un assouplissement de la structuration des transactions a été étudié par [FRANK 82].

5.3 - Utilisation de transactions emboitées Consuelo Franky-Toro. Thèse 3ème cycle [FRANK 82]

Le développement de l'informatique répartie peut s'expliquer par l'abaissement du coût des matériels et par un changement d'objectif : on cherche plus à favoriser la disponibilité des données que la puissance de calcul. Dans le domaine des bases de données, la répartition des données et des traitements complique le problème de contrôle de concurrence : les contrôleurs chargés de régler les conflits d'accès aux ressources partagées doivent prendre des décisions à partir de vues partielles et différentes de la situation. La thèse de Franky-Toro [FRANK 82] propose un mécanisme de contrôle réparti, dans le cadre des bases de données partitionnées, qui favorise l'utilisateur interactif. En utilisant un verrouillage sélectif et des transactions imbriquées, ce mécanisme permet d'augmenter la rapidité des réponses à l'utilisateur ayant commencé une transaction tout en diminuant la rigidité du traitement : l'utilisateur pourra demander une annulation partielle de la transaction. Le verrouillage sélectif consiste à proposer au programmeur différents modes de verrous (différents privilèges : lecture seule, lecture pour mise à jour, intention d'écriture ...) pouvant s'appliquer à différentes tailles de "granules" : enregistrements, segments, fichiers ... Ceci permet, par exemple, de n'accaparer à chaque instant que l'enregistrement à mettre à jour tout en déclarant, au niveau du fichier, l'intention de modifier plusieurs enregistrements. Mais il faut trouver un compromis entre la surcharge de gestion des verrous et le gain en concurrence et en disponibilité des données.

Le concept de transaction emboitée permet d'utiliser ces possibilités : les ressources globales seront réservées dans des blocs externes tandis que les ressources ponctuelles seront déclarées dans les blocs plus internes. Une transaction complexe n'est plus une suite de transactions élémentaires à deux phases (réservation de toutes les données nécessaires puis libération en bloc), mais la conformité du modèle de transactions emboitées a été prouvée.

Ce mécanisme évite donc les attentes intermédiaires sur les données globales. Pour ne pas gêner d'autres utilisateurs éventuels, les données réservées dans les blocs internes peuvent être libérées à la fin de ces blocs, qui seront considérés comme des unités de cohérence. Ceci permet de valider partiellement une transaction complexe, et d'éviter l'annulation complète pour cause de panne ou de risque d'interblocage. Mais toute validation, même graduelle, entraîne l'impossibilité de reprise en amont du point de libération (risque d'"effet domino"). Pour conserver une véritable hiérarchie des points de reprise, le programmeur dispose d'une fin de bloc sans libération. Les traitements concernant les données partagées (liste d'intention) ne sont pas réalisés mais incorporés à ceux du bloc englobant (gestion en pile). L'augmentation des possibilités de reprise sera alors payée par une moins grande disponibilité des données pour les autres utilisateurs.

Enfin, une proposition de verrous généralisés est évoquée : les verrous et leur hiérarchie seraient définis pour chaque type d'objet, ce qui correspond à la notion de type abstrait (on définit un ensemble de valeurs et les opérations qui sont permises sur ces valeurs).

Cette étude montre donc la faisabilité d'une gestion en pile des données partagées liée à la structuration des transactions, et ceci en contexte réparti. Bien que la disponibilité des données réservées dans les blocs englobants soit augmentée au détriment des autres utilisateurs, il est probable que ce schéma augmentera le "taux de service" de l'ensemble des utilisateurs : les transactions commencées pourront être menées à terme plus rapidement, et les annulations pour cause de panne ou d'interblocage seront limitées. Le programmeur devra cependant veiller à ne pas écrire des transactions trop "gourmandes".

Par rapport au modèle de l'interaction arbre statique/arbre dynamique, ce travail donne la possibilité d'assouplir considérablement l'arbre statique ; le schéma traditionnel des transactions à deux phases ne permet en effet qu'une juxtaposition de blocs sans aucune imbrication.

Conclusion

Ces trois exemples montrent donc le profit que l'on peut tirer d'un modèle d'interaction basé sur des arborescences, dans des domaines aussi différents que le parallélisme, l'infographie interactive et les bases de données réparties. Plus qu'une simple structure de données, l'arborescence devient le schéma d'une application, dont l'exploitation permet d'assouplir considérablement les contraintes d'exécution.

Le langage de dialogue proposé dans cette thèse n'est que la continuité de l'ensemble de ces travaux : les entrées-sorties de SAUGE évoquent explicitement le modèle arbre statique/arbre dynamique, le contrôle de cohérence proposé par Franky-Toro résoud le problème de gestion en pile des données partagées et l'éditeur graphique montre les possibilités données par l'association d'attributs de mise en écran aux noeuds d'une arborescence.

Enfin, les outils théoriques de manipulation d'arbres qui se développent actuellement (transducteurs d'arbres ...) laissent espérer que les arborescences prendront une importance considérable dans l'informatique de demain.

CONCLUSION

L'exigence des utilisateurs d'applications interactives, qui désirent un dialogue très souple, est souvent incompatible avec le coût de programmation accepté.

Les propositions de cette étude permettent de concilier la souplesse d'exécution et la simplicité de programmation en rendant implicite la gestion des reprises.

Ceci est possible grâce à la création, à la compilation, de blocs de reprise imbriqués dans les blocs du programme et à la gestion, à l'exécution, d'une trace du dialogue et d'un écran virtuel.

Les mécanismes proposés s'inspirent d'un modèle de l'interaction basé sur des arborescences, qui entre donc dans le cadre plus général de la programmation dite structurée.

Ce travail pourrait se poursuivre suivant deux axes différents :
- une implémentation réelle du langage, qui permettrait de vérifier l'adéquation des choix envisagés
- une étude plus approfondie des modèles de l'interaction.

En particulier, la confrontation de différents modèles avec les nouvelles techniques de l'interaction (paroles, images ...) pourrait se révéler féconde.

BIBLIOGRAPHIE

[ATTAR 81] G. ATTARDI, M. SUNI
"The Power of Programming by Examples"
Actes Congrès INRIA-KAYAK St Maximin 1981 [KAYA 81].

[BLOIS 81] J.P. BOIS
"Le Dialogue Homme-Machine en bureautique, Conception et Mise en oeuvre des interfaces"
Afcet Sicob bureautique Actes du Congrès mai 1981, Paris.

[BOTTER 82] J.M. BOTTERHILL
"The design rationale of the System/38 User Interface"
IBM System Journal Vol 21 n° 4 1982

[CACM 83]
"Working toward successful human-computer interface"
Communication of the ACM 26, n° 4 (Avril 1983).

[CARRE 80] C. CARREZ
"Structuration des programmes de gestion dans un environnement conversationnel"
RAIRO Informatique Computer Science Vol 14 n° 2 1980

[CHEVA 79] R.J. CHEVANCE
"Dialogue homme/machine en bureautique"
"office of future" North Holland 1979.

[CODD 70] E.F. CODD
"A relational model for large shared data banks"
Communication of the A.C.M. 13 n° 6 377-387 (1970)

[CORNA 81] Groupe CORNAFION
"Les système Informatiques Répartis : Concepts et Techniques"
Dunod Ed ; 1981.

[COURT 73] J. COURTIN
"Un analyseur syntaxique interactif pour la communication homme/machine"
Proceedings ICCL Pise 73.

[CROCU 75] Collectif CROCUS
"Systèmes d'exploitation des Ordinateurs"
Dunod Phase spécialité informatique (1975)

[DAUCH 77] M. DAUCHET
"Transductions de forêts. Bimorphismes de Magmoïdes"
Thèse d'Etat Université de Lille 1977.

[DEAN 82] M. DEAN
"How a Computer should talk to people"
IBM System Journal Vol 21 n° 4 1982

[DFC] Displey Formatting and Control
"Manuel d'utilisation de l'utilitaire DFC"
CII Honeywell Bull

[DOST 77] M. DOSTATNI
"Programmes dirigés par les données d'entrée"
Bulletin Groplan, AFCET, n° 2, 1977, p 31-40.

[DRIEUX] B. DRIEUX
"Le langage LOGTRAN"
SERLIE Rapport Interne

[DUTHI 82] A.M. DUTHILLEUL, G. GRIMONPREZ
"Logiciel de gestion de grilles d'écran"
IUT Informatique Lille Avril 82.

[DUYER 81] B. DUYER
"A User Friendly Algorithm"
Communications of the ACM 24,9 pp 556-561 Sept. 1981.

[FAULL 80] B. FAULLE, G. THOMAS
"L'analyse des Systèmes Conversationnels"
Zéro-Un Informatique, n° 137, Janvier-Février 1980.

[FRANK 82] C. FRANKY-TORO
"Contribution au contrôle de cohérence des systèmes transactionnels répartis en vue de favoriser l'utilisateur interactif".
Thèse 3ème cycle Université de Lille Janvier 82.

[GEHAN 83] N.H. GEHANI
"High Level Form Definition In Office Information System"
Computer Journal Vol 26 n° 1. Février 1983.

[GUEDJ 80] GUEDJ R.A.
"Methodology of Interaction"
North Holland 1980.

[HAYES 81] P. HAYES, E. BALL, R. REDELY
"Breaking the Man-Machine Communication Barrier"
IEEE Computer, Vol 14 n° 3 pp 19-30 March 1981.

[ICHBI 79] J.D. ICHBICH, J.C. HELIARD, O. ROUBINE, J.G.P. BOUNE, B. KRIEG-BRUEDINER, B.A. WICHMANN
"Reference Manual for the ADA programming language"
SIGPLAN NOTICES, Vol 14, n° 6, part A, June 1979.

[JAMES 80] E.B. JAMES
"The user interface"
The Computer Journal, Vol 23, n° 1, pp 25-28, Feb. 1980.

[KAIS 82] P. KAISER, I. STETURA
"A dialogue Generator"
Software Practice & experience Vol 12 693-707 (1982).

[KAYA 81] Projet Kayak
"International Workshop on Office Information Systems"
Congrès INRIA octobre 1981 Saint Maximin.

[LAFUE 78] J.M. LAFUENTE, D. GRIES
"Language facilities for programming user-computer dialogues"
IBM Journal of Research & Developpment Vol 22 n° 2 pp 145-158
March 1978.

[LANTZ 81] K.A. LANTZ
"Uniform Interfaces for Distributed Systems"
Thesis University of Rochester 1980.

[LECOU 82] M.P. LECOUFFE, P. LECOUFFE, J.C. MARTI, B. TOURSEL,
"Entrées-Sorties non liées au mode de séquencement : application aux Machines parallèles"
Rapport intermédiaire contrat CNET ATP FCS décembre 1982.

[LEDGA 80] H. LEDGARD, J.A. WHITESIDE, A. SINGEOR, W. SEYMOUR,
"The Natural Language of Interactive Systems"
Communication of the ACM 23, 10, pp 556-563, octobre 1980.

[LISKO 74] B. LISKOV, S. ZILLES
"Programming with abstract data types"
ACM SIGPLAN, Vol 9, n° 4, pp 50-59, Avril 1974.

[LUM 81] V. LUM, N.C. SHU, F. TUNG, C.L. CHANG,
"Automatic Business Procedures With Form Processing"
Actes Congrès INRIA KAYAK St Maximin 1981 ([KAYA 81]).

[MACCH 79] C. MACCHI, J.F. GUILBERT,
"Téléinformatique"
DUNOD, Ed. 1979.

[MARTI 73] J. MARTIN
"Design of Man-Computer dialogues"
Prentice Hall, Englewood Cliffs, New-Jersey, 1973.

[MEYER 82] B. MEYER
"Environnement Conversationnel à deux dimensions"
Actes des Journées BIGRE 82 Grenoble Janvier 1982.

[MR 80] *Reference Manual for the ADA programming language*
DOD, The Pentagon Washington DC 2030 1980.

[NEWMA 80] NEWMAN
"Some notes on User Interface Design"
Methodology of Interaction North Holland 1980.

[OUDIN 80] J. OUDIN
"Conception et Implémentation d'applications interactives en contexte industriel"
Thèse docteur Ingénieur Lille Juin 1980.

[PAREN 82] C. PARENT
"Etude et Spécifications d'un éditeur d'images, Applications à la bureautique"
Thèse 3ème cycle Lille Octobre 1982.

[PERK 80] T. E. PERKINS
"A nonprocedural language for the specification and design of Business data processing systems"
Thèse PhD Southern Methodist University Juillet 1980.

[ROWE 83] L.A. ROWE, K.A. SHOENS
"Programming Language Constructs for Screen Definition"
IEEE Transaction on software engineering Vol SE 8 n° 1 Janvier 1983

[SHU 82] N.C. SHU, V.Y. LUM, F.C. TUNG, C.L. CHANG,
"Spécification of forms Processing and Business Procedures for Office Automation"
IEEE Transactions on software engineering, Vol SE 8, n° 5, Septembre 1982

[SIGPL 74] *"Proceeding of a Symposium on Very High Level Languages"*
SIGPLAN NOTICES Vol 9, n° 4, Avril 1974.

[STERL 74] T.D. STERLING
"Guidelines for humanizing computerized Information Systems : a report from Stanley House"
CACM, Vol 7, n° 11, pp 609-613, Nov. 1974.

[UHLIG 79] R.P. UHLIG, D.J. FARBER, J.M. BAIR,
"The Office of future"
North Holland 1979.

[VERRA 82] D. LE VERRAND (nom collectif)
"Le langage ADA manuel d'évaluation"
DUNOD 1982.

[WINOG 79] T. WINOGRAD
"Beyond Programming Languages",
CACM, Vol 22, n° 7, July 1979.

[ZLOOF 76] M.M. ZLOOF, S.P. DEJONG
"The System for Businesse Automation (SBA)"
Communication of the ACM 20, n° 6, 385-396 (1977)

[ZLOOF 77] M. ZLOOF
"Query by-Example : A data base language"
IBM Systems Journal 16, n° 4, 324-343 (1977)

[ZLOOF 82] M.M. ZLOOF
"Office by-Example : A business language that unifies data and word processing and electronic mail"
IBM System Journal Vol 21 n° 3, 1982.

ANNEXE :

EXEMPLE DEVELOPPE

I - DÉROULEMENT DU DIALOGUE

I - Première grille du dialogue :

```
Que voulez-vous faire ;
  * Création
  * Modification
  * Suppression
```

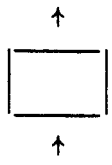
Utilisateur confirmé : Concision des messages.
Choix par déplacement du curseur (ici : création).

II -

```
Que voulez-vous faire ;
  * Création
  * Modification
  * Suppression
Nom du modèle ? : Blouson
Paramètres ? : Tailles Enfant
```

Apparition de la sous-grille de saisie du modèle ; le contrôle syntaxique se fait en temps réel, mais le "processeur de traitement" ne prendra connaissance de la valeur des champs qu'en fin de sous-grille.

III - Apparition de la sous-grille de saisie des produits ; disparition, faute de place, de la sous-grille la plus ancienne : défilement "en rouleau" au niveau de l'écran entier :



```
Nom du modèle ? : Blouson
Paramètres ? : Tailles Enfant
1ième produit du modèle Blouson
Matériau 1 : ...
Matériau 2 : ...
```

IV - L'utilisateur entre alors les matériaux demandés :

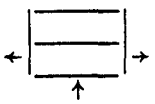
N'utilisez que des lettres
Nom du modèle ? : Blouson
Paramètres ? : Tailles Enfant
1ième produit du modèle Blouson
Matériau 1 : Coto 7
Matériau 2 :

La "ligne système" est utilisée pour l'affichage des messages d'erreur.

V - Fin de saisie du premier produit attaché au modèle Blouson

Nom du modèle ? : Blouson
Paramètres ? : Tailles Enfant
1ième produit du modèle Blouson
Matériau 1 : Coton
Matériau 2 : Cuir

VI - La sous-grille suivante correspond à la saisie du second produit du modèle. Pour éviter de priver l'utilisateur du contexte lié au modèle, le programmeur a choisi un défilement en rouleau à l'intérieur d'un sous-écran :



Nom du modèle ? : Blouson
Paramètres ? : Tailles Enfant
2iéme produit du modèle Blouson
Matériau 1 : ...
Matériau 2 : ...

VII - Par déplacement arrière du curseur, l'utilisateur peut visualiser un contexte ancien ayant disparu faute de place.

Nom du modèle ? : Blouson
Paramètres ? : Tailles Enfant
1ième produit du modèle Blouson
Matériau 1 : Coton
Matériau 2 : Cuir

"Rouleau inverse" dans un sous-écran.

VIII - "Rouleau inverse" dans l'écran

Que voulez-vous faire ? :
* Création
* Modification
* Suppression
Nom du modèle ? : Blouson
Paramètres ? : Tailles Enfant

IX - Lorsque l'utilisateur modifie un champ en saisie d'une sous-grille déjà terminée (ici, les enfants grandissent) une reprise est provoquée :

Que voulez-vous faire ? :
* Création
* Modification
* Suppression
Nom du modèle ? : Blouson
Paramètres : Tailles Adulte

X - Les saisies postérieures à la donnée modifiée et qui restent cohérentes sont réutilisées sous réserve de confirmation.

Modification ?
Nom du modèle ? : Blouson
Paramètres : Taille Adulte
1ième Produit du modèle Blouson
Matériau 1 : Coton
Matériau 2 : Cuir

XI - Fin de la reprise, le dialogue continue

Nom du modèle ? : Blouson Paramètres : Tailles Adulte
2ième produit du modèle Blouson Matériau 1 : ... Matériau 2 : ...

XII - Une touche "aide" permet d'afficher un commentaire spécifique au point courant du programme (ici : le produit)

Touche Aide ou Retour
Chaque produit est défini par deux de ses matériaux, qui ne doivent comporter que des lettres. L'entrée vide signale que la liste des produits est terminée.

XIII - Un nouvel appui sur la touche aide entraine l'affichage d'un commentaire plus général (ici : le modèle).

Touche Aide ou Retour
Le nom d'un modèle comporte au plus vingt lettres, ses paramètres ne sont pas limités. A un modèle sera attaché un ou plusieurs produits. L'entrée vide indique la fin de création des modèles.

XIV - Retour au dialogue, en repassant par les aides intermédiaire

Nom du modèle ? : Blouson Paramètres ? : Tailles Adulte
2ième produit du modèle Blouson Matériau 1 : ... Matériau 2 : ...

L'utilisateur signale par l'entrée vide qu'il a terminé la création des produits du modèle blouson.

XV - Le système lui demande alors confirmation de toutes les entrées : il y a validation en fin de modèle, donc interdiction de reprise

Vérifiez les saisies avant validation
Nom du modèle ? : Blouson
Paramètres ? : Taille Adulte
1ième produit du modèle Blouson
Matériau 1 : Coton
Matériau 2 : Cuir

L'utilisateur pourra ensuite créer un autre modèle ...

Les deux caractéristiques essentielles sont le déroulement hiérarchique du dialogue et la grande facilité de reprises.

Le déroulement hiérarchique permet à l'utilisateur de passer du général (l'application dans son ensemble) au particulier (sous-grille de saisie du produit ...). Les aides qui lui sont fournies tiennent compte du "niveau de profondeur" où il se trouve. Cette hiérarchie permet également l'affichage d'un contexte plus vaste en limitant l'encombrement sur écran des niveaux de détail. L'utilisateur peut "cheminer" simplement dans la portion du dialogue qu'ila déjà achevée : cheminement dans les aides ou retour sur des grilles effacées.

C'est cette possibilité de retour du curseur sur tout champ de saisie même effacé qui va entrainer la facilité des reprises ; l'utilisateur peut modifier une saisie déjà acceptée par le système. Une reprise est alors déclenchée, avec réutilisation implicite de toutes les saisies antérieures à la saisie modifiée et réutilisation sous réserve de confirmation de toutes les données postérieures encore cohérentes.

II - PROGRAMMATION DU DIALOGUE

```
procédure  produits - textiles
début type transaction : ensemble (CREATION, MODIFICATION, SUPPRESSION) ;
      type modèle : [1..20] de lettres
      type matériau : alphabétique
      saisir (trans : transaction) ;
      cas trans dans
          CREATION : répéter jusqu'à entrée vide
          début      saisir (mod : modèle) ;
                     saisir (paramètres) ;
                     contrôler (mod ∉ liste modèles) ;
                     I : entier := 0
              répéter jusqu'à entrée vide
          début I := I+1 ; saisir (produit : [1.2] de matériau) fin rémanent
          valider
      fin ;
      MODIFICATION : début ... fin ;
      SUPPRESSION :  début ... fin
fin
```

A ce stade, le programmeur a défini le déroulement "classique" du programme : il ne s'est préoccupé ni de la mise en écran, ni des possibilités de reprise, cette procédure n'a donc pas présenté plus de difficultés d'écriture qu'une procédure non conversationnelle.

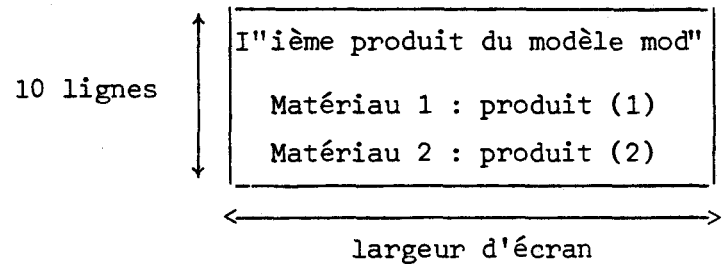
Un éditeur d'écran va ensuite aider le programmeur à définir la mise en écran de la procédure.

Cette mise en écran présente trois aspects :

- définition des messages associés aux rubriques et aux erreurs
- définition des fenêtres ("sous-écran") associées aux rubriques, sous-grilles et blocs de programme.
- définition des aides, commentaires associés aux blocs de programme.

Par exemple :

- la rubrique de saisie associée au produit sera :



Ceci nécessite la connaissance des variables accessibles en ce point du programme.

- Le sous-écran réservé à la boucle sur les produits sera de la taille de la fenêtre associée à un produit.

- A ce bloc "produit" sera associé le commentaire :

"Chaque produit est défini par deux de ses matériaux, qui ne doivent comporter que des lettres. L'entrée vide signale que la liste des produits est terminée."

Le programmeur pourra éditer un texte mixte (programme et mise en écran), ce qui donnera un programme très documenté.

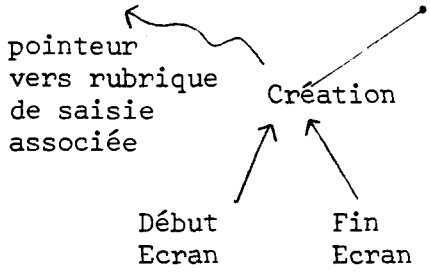
La gestion des reprises va nécessiter un double mécanisme : création, à la compilation, d'un point de reprise par instruction de saisie et gestion à l'exécution d'une trace du dialogue.

Pour la gestion de l'affichage, une seconde structure sera nécessaire : l'écran virtuel.

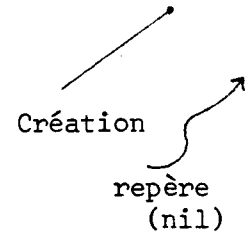
III - EVOLUTION DE L'ÉCRAN VIRTUEL ET DE LA TRACE DU DIALOGUE AU COURS DE L'EXÉCUTION

(Pour rappel des différences de ces structures, voir page 89).
Remarque : le modèle et ses paramètres sont regroupés en une sous-grille (pas d'instructions intermédiaires). Il n'y a donc qu'un point de reprise associé. Le chiffre romain correspond à l'étape du dialogue. L'état des structures correspond à la fin de l'étape.

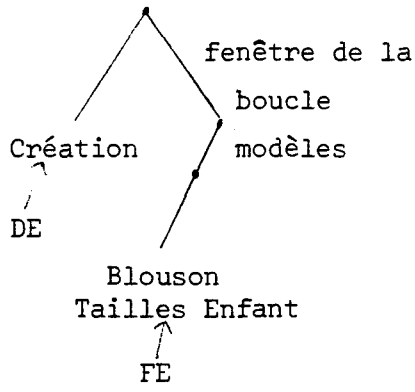
I - ECRAN VIRTUEL

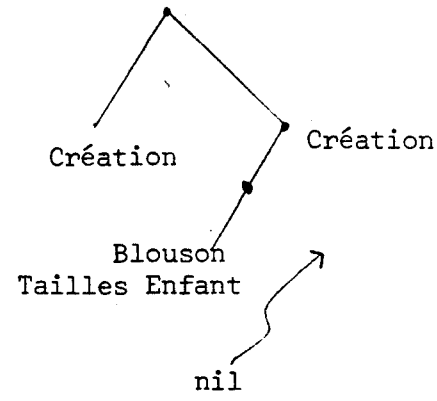


TRACE



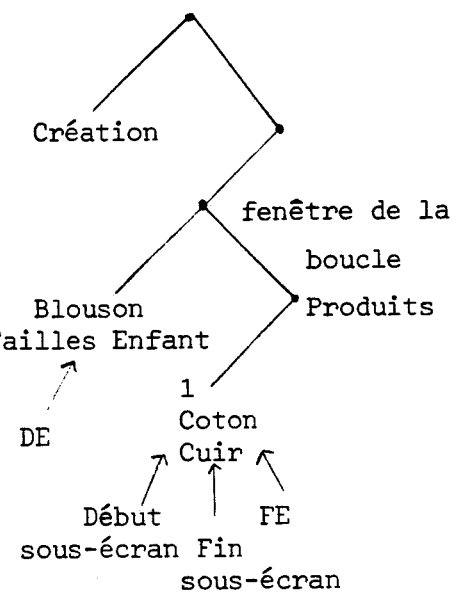
II

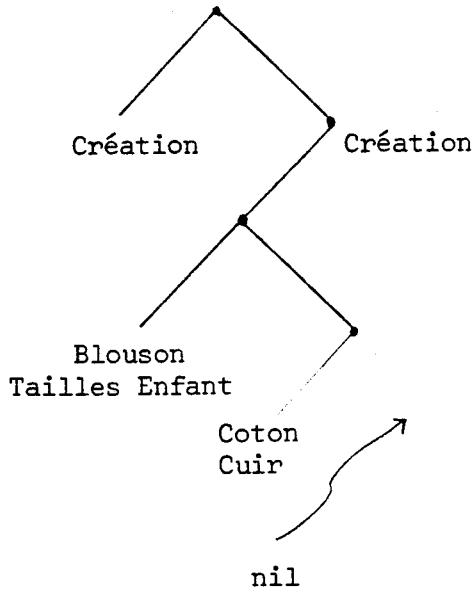




On associe la valeur de la variable de commande trans au sous-arbre correspondant

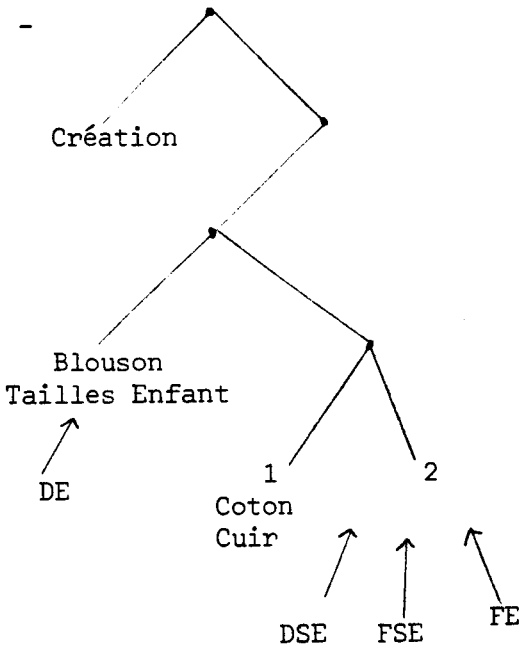
V -

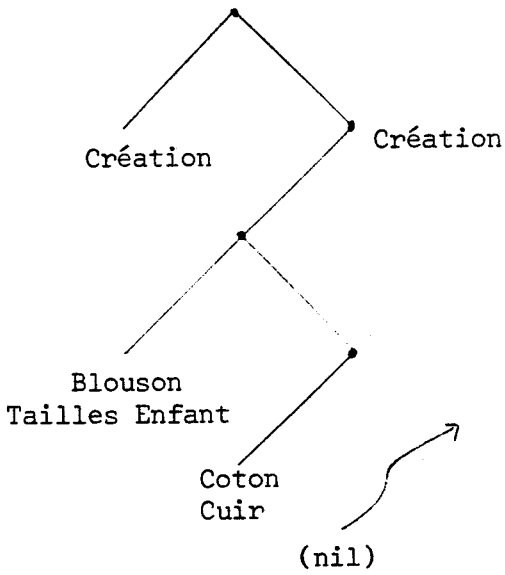




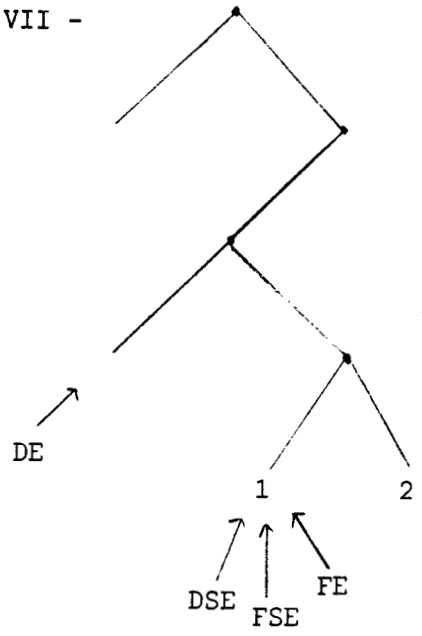
2ème niveau de "rouleau"

VI -

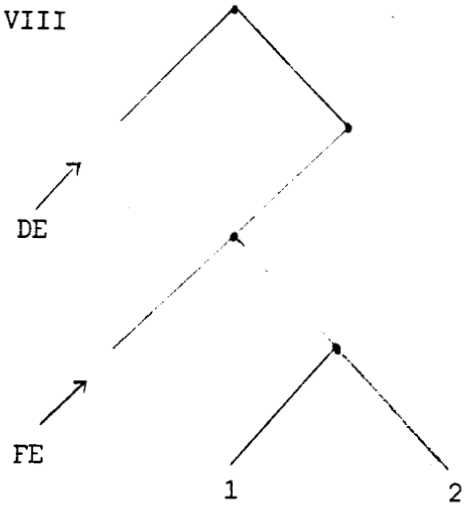


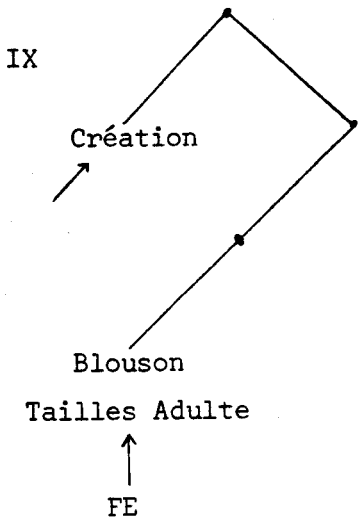


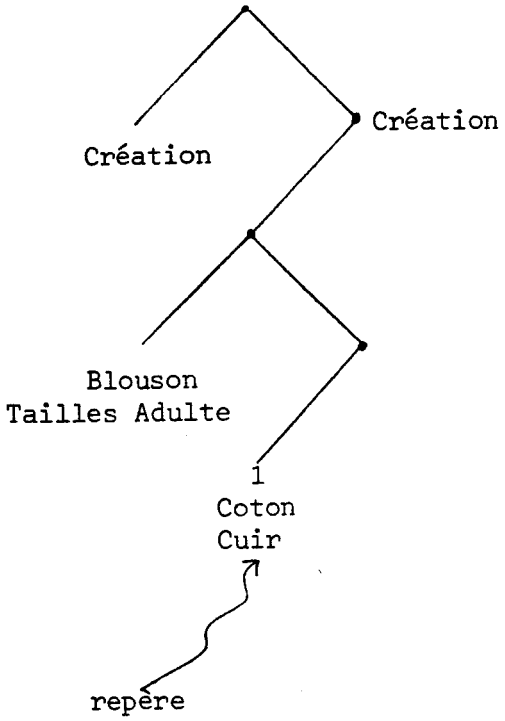
VII -



VIII







Le repère n'est plus "nil" : réutilisation des saisies.

Après la reprise, toutes les données restent cohérentes : il n'y a donc pas de suppression d'un sous-arbre de la trace.

Résumé

La diffusion des matériels et logiciels informatiques oblige à prendre en compte la psychologie et les besoins des utilisateurs non informaticiens. Mais l'écriture des interfaces permettant des "dialogues" satisfaisants devient très vite rédhibitoire. La principale difficulté dans la définition de programmes interactifs vient de la nécessité de prévoir des déroutements liés au mode conversationnel : reprise sur une phase antérieure avec annulation de traitements, demande d'aide-mémoire ...

Cette thèse développe un langage et des mécanismes qui soulageront considérablement l'effort de programmation de dialogues en gérant automatiquement ces déroutements.

Les mécanismes proposés vont exiger du programmeur qu'il s'attache à structurer convenablement son application et à définir les structures de données nécessaires et leurs types ainsi que les contraintes sémantiques sur les données. La création des rubriques de saisie ("mise en écran" des variables échangées) se fera à l'aide d'un éditeur interactif. A l'exécution, la structuration en blocs règlera l'affichage de ces rubriques.

Les reprises seront possibles grâce à la création, à la compilation, de points de sauvegarde de contexte hiérarchisés et à la gestion, à l'exécution, de structures arborescentes : trace du dialogue et écran virtuel.

Une implémentation minimale sera étudiée, ainsi que diverses extensions et une intégration du langage dans un environnement ADA. Enfin, un modèle arborescent de l'intéraction sera proposé, et différents travaux réalisés à l'Université de Lille sur le thème de l'interaction y seront confrontés.

Mots clés

intéraction, interface homme/machine ;

erreurs humaines ; reprises ;

écran virtuel ; grilles d'écran