Contribution à la modélisation de l'interface homme-machine dans un système de C.A.O

Walter Totino

Laboratoire de Recherche en Informatique de Metz

THESE

présentée à

L'UNIVERSITE DE METZ

pour l'obtention du grade de

DOCTEUR DE L'UNIVERSITE DE METZ

(mention Sciences)

SPECIALITE INFORMATIQUE

par

Walter TOTINO

" CONTRIBUTION A LA MODELISATION DE L'INTERFACE HOMME-MACHINE DANS UN SYSTEME DE C.A.O. "

Soutenue le 6 octobre 1989 devant la commission d'examen

Messieurs

Y. GARDAN (Directeur de thèse), Professeur à l'Université de Metz

P. GEORGES (Examinateur), Directeur Technique - Société ARM Conseil

G. GOVAERT (Examinateur), Professeur à l'Université de Metz

M. LUCAS (Rapporteur), Professeur à l'Ecole Nationale Supérieure de Mécanique de Nantes

R. SOENEN (Rapporteur), Professeur à l'Université de Valenciennes

" ...

Ma chi te cride d'essere... nu ddio?
Ccà dinto, 'o vvuò capì, ca simmo eguale?..
... Muorto si' tu e muorto so' pur'io;
ognuno comme a n'ato è tale e qquale ".

...

" Tu qua' Natale... Pasca e Ppifania!!!
T''o vvuò mettere 'ncapo... 'int''a cervella
che staje malato ancora 'e fantasia?...
'A morte 'o ssaje ched'è?... è una livella.

'Nu rre, 'nu maggistrato, 'nu grand'ommo
trasenno stu canciello ha fatt''o punto
c'ha perzo tutto, 'a vita e pure 'o nomme:
tu nun t'hê fatto ancora chistu cunto?

... "

Antonio De Curtis (Totò).

A mes parents.

REMERCIEMENTS

Ce travail a été réalisé au Laboratoire de Recherche en Informatique de Metz, sous la direction de Monsieur le Professeur Y. GARDAN. Je tiens à lui exprimer toute ma reconnaissance pour la confiance qu'il m'a témoignée, pour son aide, son dynamisme et sa compétence dont il m'a fait profiter tout au long de cette thèse.

Qu'il me soit permis de remercier Messieurs M. LUCAS, Professeur à l'Université de Nantes, et R. SOENEN, Professeur à l'Université de Valenciennes, pour avoir accepté de rapporter ce travail, pour les propositions judicieuses qu'ils m'ont suggérées et pour leur participation à la commission d'examen.

Mes remerciements vont également à Messieurs G. GOVAERT, Professeur à l'Université de Metz et P. GEORGES, Directeur Adjoint de la société ARM Conseil, pour avoir accepté de juger mon travail.

J'exprime ma gratitude à la société A.R.M. Conseil et à l'Association Nationale de la Recherche Technique (A.N.R.T.) pour leur soutien financier.

Je suis très reconnaissant à Isabelle, membre de l'équipe L.R.I.M., pour l'intérêt qu'elle a porté à mon travail et pour avoir assuré une lecture attentive du manuscrit.

Je ne saurais oublier les autres membres du laboratoire qui m'ont aidé tout au long de ma présence parmi eux, que ce soit par leurs compétences respectives ou par leur amitié.

Enfin, j'adresse une pensée particulière à Dominique, Jocelyne et Martine, techniciennes du laboratoire, pour leur gentillesse et leur disponibilité à toute épreuve.

TABLE DES MATIERES

INTRODUCTION 11

PREMIERE PARTIE : *Des bases pour une réponse à l'opérateur*

I.1. OBJECTIFS DU SYSTEME SACADO 15

I.1.1. CONTRAINTES MATERIELLES 15
I.1.2. CONTRAINTES INDUSTRIELLES 15
I.1.3. CONTRAINTES THEORIQUES 16

I.2. BASES DE L'INTERFACE 16

I.2.1. OBJECTIFS 16
I.2.2. EXIGENCES 17
I.2.3. PRINCIPE 18

I.3. UNE REPONSE A L'UTILISATEUR FINAL 19

I.3.1. CONFIGURATION MATERIELLE 19
I.3.2. FONCTIONNEMENT 19
I.3.3. L'IDEE CENTRALE 20
I.3.4. TYPES DE MENUS 21
I.3.5. CONVENTIONS 23

DEUXIEME PARTIE : *Architecture logicielle*

II.1. INTRODUCTION 29

II.2. ORGANISATION DES DIFFERENTS MODELES 29

II.3. LE MODELE GENERIQUE 30

II.4. LE MODELE DE VISUALISATION 31

II.4.1. LE PROBLEME ELEMENTAIRE 32
II.4.2. GENERALISATION 33

II.5. LE MODELE D'INTERFACE : UNE PREMIERE VERSION 35

II 5.1. STRUCTURE DE MENUS 36
II.5.2. DECLENCHEMENT PROCEDURAL 39
II.5.3. RESUME 45

TROISIEME PARTIE : *Le modèle complet de l'interface*

III.1. AMELIORATION DU MODELE 49

III.1.1. CARACTERISATION D'UNE INTENTION..... 49
III.1.2. CARACTERISATION D'UNE ACTION 52

III.2. DEUXIEME VERSION 58

III.2.1. PLACE DE L'INTERACTION DANS LE MODELE 58
III.2.2. CHOIX LOGICIELS 63

III.3. "DERNIERE" VERSION 67

III.3.1. STRUCTURE COMPLEMENTAIRE 67
III.3.2. OPTIONS PREDEFINIES 70
III.3.3. MECANISMES PREDEFINIS 73
III.3.4. REVISIONS DES PRIMITIVES 74

III.4. SCHEMA GENERAL 77

QUATRIEME PARTIE : *Deux utilisateurs privilégiés*

IV.1. UNE REPONSE AU CONCEPTEUR 81

IV.2. UNE REPONSE AU PROGRAMMEUR 83

IV.2.1. SPECIFICATION DE LA SEMANTIQUE D'UNE ACTION 84
IV.2.2. UTILISATION DE LA SPECIFICATION D'UNE ACTION 86

CINQUIEME PARTIE : *Autres modèles et autres systèmes*

V.1. LE MODELE LINGUISTIQUE 91

V.2. LE MODELE EVENEMENT 94

V.3. MODELES ORIGINAUX 98

CONCLUSION 109

ANNEXE 1 : Exemples de fonctionnalités de SACADO 113

ANNEXE 2 : Schémas du modèle de visualisation de SACADO 123
Spécification de l'environnement.
Exemples de scènes dans l'environnement multi-fenêtrage

ANNEXE 3 : L'écran de l'application "concepteur d'interface" 137

BIBLIOGRAPHIE 147

INTRODUCTION

Les 20 dernières années ont donné naissance à d'innombrables systèmes graphiques interactifs. Ceux-ci ont évolué avec l'accroissement des performances du matériel, d'une part, et avec l'affinement et la formalisation des différentes approches, d'autre part. Le souci de chacun étant d'établir les bases des interfaces répondant au mieux aux exigences des utilisateurs.

Le problème est ouvert dans la mesure où il est encore mal défini. En effet, contrairement à la gestion des Bases de Données, par exemple, les critères de qualité sont loin d'être acceptés par l'ensemble de la communauté concernée. En fait, on s'aperçoit que la multitude des idées donne parfois lieu à des approches contradictoires, tout au moins dans les solutions proposées.

C'est dans ce cadre que l'équipe C.A.O. et I.A. du L.R.I.M. tente d'apporter sa contribution au développement de tels systèmes. Nous allons décrire notre démarche en présentant nos objectifs, ainsi que les solutions que nous proposons. Au travers de notre expérience, nous dégagerons les concepts que nous entendons défendre tout en posant les problèmes rencontrés. Ceci nous permettra de nous situer par rapport à d'autres travaux, ainsi que de faire le point sur l'état d'avancement de notre projet.

Une première partie donne des points très généraux concernant toute l'équipe. Ce sera l'occasion, après avoir fixé des objectifs d'ensemble, de brièvement présenter le système SACADO (Système Adaptatif de Conception Assistée et de Développement par Ordinateur) d'un point de vue externe. Nous en profiterons pour introduire la terminologie et les conventions utiles pour la suite de l'exposé.

Après quoi, nous situerons l'interface homme-machine comme faisant partie d'une série de modèles qui font l'objet de la deuxième partie.

Nous nous intéressons plus particulièrement aux aspects dialogue de SACADO. C'est pourquoi, nous proposons un modèle d'interface autour duquel s'articule l'architecture logicielle. La troisième partie est constituée de raffinements successifs (et progressifs) du modèle de dialogue.

Les soucis (et la nécessité) d'ouverture et de convivialité accrues conduisent à l'intégration d'applications particulières qui visent à renforcer le caractère Adaptatif et les capacités de Développement Assisté de SACADO. Nos travaux actuels et nos orientations futures à ce sujet sont présentés dans la quatrième partie.

Ce n'est qu'à partir de là qu'il nous semble raisonnable de comparer notre approche à d'autres modèles. Ainsi, une cinquième partie nous permettra de situer SACADO par rapport à d'autres systèmes et de donner quelques formalismes rencontrés dans la littérature.

Nous concluerons par les travaux à venir, tant en fonction de la présente étude, que par rapport à l'ensemble de l'équipe.

PREMIERE PARTIE

DES BASES POUR UNE REPONSE A L'OPERATEUR

Cette première partie vise essentiellement à présenter les grandes orientations de l'équipe. Après un rapide aperçu des travaux du L.R.I.M., nous énonçons les principes que nous entendons défendre en matière d'interface homme-machine. Un exposé succinct des possibilités offertes à un opérateur devrait mettre en évidence la philosophie du dialogue dans SACADO.

La terminologie introduite dans les paragraphes suivants donne une idée des concepts caractéristiques de notre approche. De même, les exemples d'utilisation des organes d'entrées nous semblent révélateurs des capacités du système.

I.1 OBJECTIFS DU SYSTEME SACADO

Il nous a semblé primordial de développer une application C.A.O. réaliste, afin d'être confrontés à des problèmes pratiques tout en menant une réflexion théorique. Dès le départ nous nous sommes volontairement imposé des contraintes [GAR 86a].

I.1.1. CONTRAINTES MATERIELLES

Les coûts prohibitifs des grands systèmes de C.A.O. et leur incapacité à s'adapter à des matériels plus petits, ainsi que l'essor de la microinformatique, expliquent en partie notre volonté de développer un système viable dans un environnement micro. Il est impératif de répondre à la demande croissante de "petits utilisateurs" (universités, P.M.E., etc...). De plus, ceci nous incitera à étudier avec soin le problème du compromis espace-temps afin d'optimiser les algorithmes et de choisir une architecture en tenant compte des limites du matériel.

Un premier objectif est donc la ***faisabilité*** d'un tel système dans un environnement microordinateur, en autorisant la ***portabilité*** [PEL 84] sur des machines plus conséquentes.

I.1.2. CONTRAINTES INDUSTRIELLES

Notre projet est en partie financé par l'ANVAR [ANV 89] et par une société (A.R.M. Conseil) [ARM 88] qui se chargera de son industrialisation. Ceci nous oblige à rester concrets et réalistes. Le produit final doit être concurrentiel, ce qui définit deux autres objectifs :

- le système doit être ***fonctionnel*** afin de répondre à la demande,
- le système doit être ***convivial*** afin d'être attractif.

I.1.3. CONTRAINTES THEORIQUES

Plusieurs sujets de recherche ont été définis à partir des différents composants d'un tel système [LRIM 87a]:

- modélisation géométrique [JUN 87][JUN 89][MIN 88][PUC 89a][PUC 89b]
- surfaces gauches [SAH 87][MIC 87][SAH 89][MIC 90]
- opérations booléennes [MAR 86][PUC 88]
- structures et algorithmes [PUC 87][MIN 86][MIN 88][KOL 87][KOL 88][KOL 89][KOL 90][MIN 90][VIV 90]
- dialogue et architecture logicielle [TOT 86][GAR 87][TOT 87][TOT 88][LRIM 88][TOT 89][GAR 89c]
- paramétrage [SLO 89a][SLO 89b]

Ainsi, chaque membre de l'équipe mène une réflexion spécifique qui doit s'intégrer à un tout. La ***modularité*** et l'approfondissement des ***aspects théoriques*** constituent des objectifs importants.

Nous venons de présenter la démarche dans son ensemble. Les points pré-cités permettent de dégager un objectif qu'il ne faut pas perdre de vue, à savoir la ***validation*** de l'approche. Dans la suite, nous nous attacherons plus particulièrement aux aspects "dialogue" du système.

I.2. BASES DE L'INTERFACE

Malgré les difficultés à définir les orientations à prendre dans le développement de systèmes graphiques interactifs, il semble de plus en plus acquis que la souplesse d'utilisation est un élément fondamental de choix de système [FOL 82]. Un effort particulier doit donc être porté sur l'***ouverture*** et la ***convivialité*** de l'interface avec l'utilisateur [GAR 83].

I.2.1. OBJECTIFS

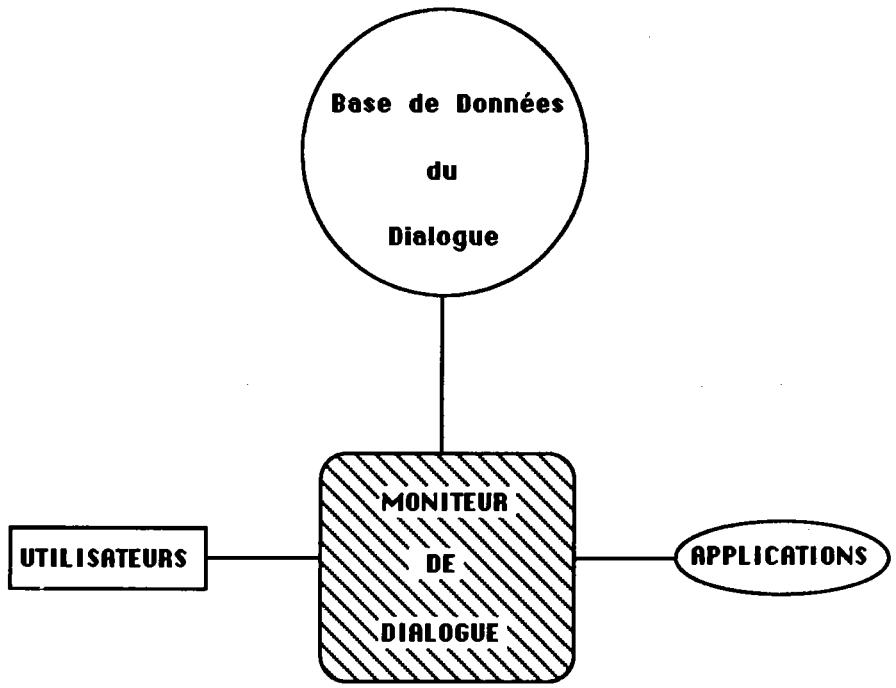
Notre but est de définir des ***principes indépendants des applications*** afin d'en garantir la généralité [LRIM 88].

Avant tout, il est indispensable de définir l'utilisateur afin de cerner ses préoccupations. Nous considérons qu'il y a ***trois types d'utilisateurs*** :

- l'utilisateur final (opérateur),
- le concepteur d'interface,
- le programmeur d'application.

L'utilisateur final travaille avec une application dont les fonctionnalités sont mises en oeuvre par le programmeur d'application, et il communique avec celle-ci grâce à un dialogue développé par le concepteur. En fait, le programmeur utilise une application qui lui permet d'écrire du code. De même, le concepteur utilise une application qui lui permet de définir une interface.

Bien que ces utilisateurs manipulent des entités de domaines différents, l'objectif dominant de notre approche est d'***intégrer leurs points de vue,*** en offrant une architecture centrée sur un dialogue "standard" (figure I.1.).



(figure I.1.)

D'ores et déjà, s'affirme une volonté d'***uniformité*** de l'outil tout en conservant une relative ***indépendance des rôles.***

C'est à partir des réponses à apporter à chacun des trois types d'utilisateurs, que nous définirons les concepts fondamentaux qui nous conduiront à l'***architecture logicielle*** en passant par la ***modélisation de l'interface.***

I.2.2. EXIGENCES

En dehors des fonctionnalités qui lui sont proposées (création d'objets, cotations, ...), ce qui intéresse un utilisateur c'est la façon dont elles lui sont présentées, comment il y accède, et quelles sont leurs relations.

Le système doit être agréable, souple, compréhensible, consistant, complet, adaptable, extensible [FOL 82][MIL 77][TRE 77][OLS 84a][GAR 87]. Tous ces points sont pris en compte par la plupart des auteurs. D'un point de vue externe, ils relèvent de l'étude des facteurs humains (ergonomie, psychologie, ...) et font l'objet d'un assez large consensus [FOL 74][BLE 82][REI 83]. D'un point de vue interne, ils conduisent à des choix (des approches) différents, tant par leur place dans la modélisation que dans la façon dont ils sont gérés.

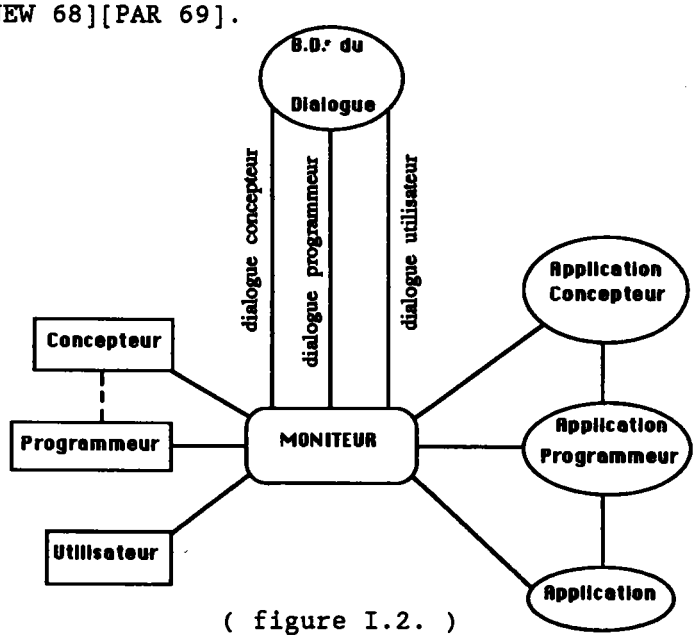
Par exemple, une interface peut être "excellente" pour l'utilisateur final. Mais si elle donne lieu à une implantation utilisant des concepts trop complexes, et surtout trop variés suivant les utilisateurs, elle risque de se refermer sur elle- même dans la mesure où elle compliquera les tâches du concepteur et du programmeur. Les conséquences en seraient :

- difficulté de mise en oeuvre
- ouverture et adaptabilité compromises
- lourdeur de la maintenance
- validation délicate

I.2.3. PRINCIPE

Il nous apparaît donc fondamental d'approcher (et donc de développer) ces 3 éléments (utilisation, interface, programmation) en parallèle. En d'autres termes, nous sommes convaincus que ces trois types d'utilisateurs doivent bénéficier d'un outil dont la puissance se caractérise par l'introduction de concepts uniques qui les servent tous. A cet égard, nous sommes en accord complet avec ROACH [ROA 82] qui propose une architecture schématisant très bien les principes évoqués jusqu'ici (figure I.2.). Bien que les rôles de chacun soient fonctionnellement séparés, les différents intervenants sont amenés à communiquer.

Ceci se traduit par la ***dualité*** qui existe entre l'***application*** et le gestionnaire de ***dialogue*** [ROA 82][SCH 85][LRIM 88] (se servent l'un de l'autre), par opposition aux modèles basés sur une gestion du flot d'entrées (sorties) entièrement externe (séparation complète et effective du code et du dialogue) [KAS 82][LIE 85][GRE 85][FLE 87] ou entièrement interne (intégration totale du dialogue dans le code) [NEW 68][PAR 69].



(figure I.2.)

La présentation de notre système nous permettra de mieux comprendre ces notions en affinant l'approche.

I.3. UNE REPONSE A L'UTILISATEUR FINAL

Nous présentons ici les grandes lignes du système C.A.O. que le L.R.I.M. développe depuis septembre 1986. Le but de ce paragraphe n'est pas d'établir une liste exhaustive des différentes fonctionnalités qu'il offre [LRIM 87b]. Nous parlerons surtout de son aspect externe.

Rappelons simplement qu'il permet de manipuler des objets 2D et 3D qui sont modélisés géométriquement afin, non seulement, de les visualiser (dessins plans, visualisations réalistes, sorties papier, ,...), mais également d'effectuer des calculs (surfaces, volumes, moments, ...) et de les utiliser dans des modèles plus complexes (extrusions, CSG, ...). Les objets ont une réalité dans un espace utilisateur (coordonnées réelles) et on en perçoit des images (au sens large) dans un espace écran (coordonnées entières).

I.3.1. CONFIGURATION MATERIELLE

La configuration est la suivante :

- une unité centrale (compatible PC, AT, PS),
- un écran graphique,
- un clavier alphanumérique (standard),
- une souris (ou tout autre unité d'entrée possédant au moins deux gâchettes de validation),
- une sortie papier (imprimante laser).

Nous reviendrons ultérieurement sur les langages et les systèmes d'exploitation employés, puisque ces éléments ne concernent pas directement l'utilisateur final.

Nous considérons que les composants ci-dessus constituent le minimum pour un poste de travail expérimental réaliste, étant bien entendu que le souci d'indépendance par rapport au matériel demeure présent.

En ce qui concerne les unités d'entrées (clavier et souris), bien qu'il ne faille pas s'y limiter exclusivement, elles correspondent malgré tout à un ***premier principe*** : l'utilisateur ne doit pas se noyer dans le matériel qui l'entoure. Le nombre d'organes d'entrée ne doit pas être excessif afin de faciliter l'apprentissage et de simplifier l'utilisation. De plus, cela garantit une continuité tactile [FOL 74] plus forte (puisque l'on limite l'obligation de passer d'une unité à l'autre).

I.3.2. FONCTIONNEMENT

Un ensemble d'options apparaît à l'écran. Ces options (icônes) figurent les fonctionnalités proposées. Notre approche est orientée menus par opposition aux systèmes orientés langages de commandes. Elle est dans la droite ligne des nouvelles générations d'interfaces [LEV 85].

A ce stade, nous voulons mettre en évidence quelques points apparemment sans importance, mais dont les implications sont partie intégrante de notre approche. Nous sommes partisans de la localisation des menus à l'écran, plutôt que sur une tablette graphique ou sur un pavé de touches de fonctions. D'une part, ce choix accroît la continuité visuelle préconisée par FOLEY et WALLACE [FOL 74] (puisque l'attention est à tout moment focalisée sur l'écran). D'autre part, il est clair que l'écran offre de plus grandes possibilités de dynamisme (position des options non figée [donc facilement reconfigurable], visualisation des seules options utiles et accessibles à un instant donné [limite les confusions et les risques d'erreur de manipulation, augmente la consistance]). Dans ce cadre, SACADO se caractérise également par des particularités dont nous verrons les incidences ultérieurement:

- l'écran n'est pas divisé en espaces dédiés du type : une zone d'affichage, une zone de menus, une zone de messages, ...

- le comportement d'un menu est lié au contexte d'utilisation. Sa signification relativement à d'autres options pourra changer et l'utilisateur devra en être averti. Ceci implique un changement d'aspect (couleur, forme, visibilité, position, ...) qui serait irréalisable avec une autre approche (une tablette graphique par exemple).

Dès lors, l'utilisateur communique avec l'application en lui précisant ses intentions par le biais des organes d'entrées. Il choisit une option associée à une action à entreprendre. L'application exécute le traitement et fournit ses résultats à l'utilisateur (calculs, affichages, ...) . L'action aura éventuellement besoin de paramètres qu'elle demandera à l'utilisateur. Ainsi s'installe un dialogue entre l'application et l'utilisateur. Lorsque l'action se sera terminée, l'utilisateur pourra spécifier une nouvelle intention.

Ceci est le schéma classique de la plupart des systèmes interactifs. Voyons ce qui nous a incité à aller un peu plus loin.

I.3.3. L'IDEE CENTRALE

Posons dès à présent un ***autre principe*** : à chaque fois que l'utilisateur est sollicité, il doit être aussi peu contraint que possible. Ceci vaut autant pour la nature et le nombre d'interventions autorisées à un instant donné, que pour les moyens dont il dispose pour s'exprimer. Les ***intentions*** de l'utilisateur doivent être ***interprétées au mieux par le système.*** Ce point se traduit par les possibilités suivantes :

- autant que faire se peut, on doit pouvoir spécifier une entrée indifféremment au clavier et/ou à la souris. A la limite, on doit pouvoir se passer de l'une ou l'autre de ces unités.

ex. : + une option (ou un objet) peut être choisie par son nom (clavier) ou pointé à l'écran (souris)

+ des coordonnées sont précisées par leurs valeurs (clavier) ou par le passage écran-espace (souris)

- l'utilisation des deux unités peut être combinée ou alternée.

ex. : + la coordonnée X au clavier, Y et Z à la souris

+ une valeur peut s'exprimer par une expression grapho-numérique [GAR 82][GAL 83] du type " V = d(A,B) - r(C) " où A,B,C sont des noms d'objets, A et C explicitement nommés (clavier) et B identifié (souris)

- le fait qu'une action attende un paramètre (objet, coordonnées, scalaire,....) n'empêche pas pour autant l'utilisateur de sélectionner un menu, bien au contraire, que ce soit pour abandonner l'action, comme pour agrandir un détail pour mieux décider, ou apporter des précisions.....

- de même, l'ordre X puis Y (puis Z) n'est pas imposé, l'utilisateur peut le redéfinir à son gré.

Ce type de fonctionnalités implique des concepts nouveaux.

I.3.4. TYPES DE MENUS

Nous venons de voir que nous souhaitons élargir le champ d'action de l'utilisateur au maximum. Il importe donc de définir des notions significatives d'une intention, afin que :

- l'utilisateur comprenne les conséquences d'un choix,
- le système interprète et contrôle au mieux ce choix.

Le paragraphe précédent a montré qu'à tout moment on peut sélectionner un menu. Ceci confère un rôle privilégié à un menu et nous a incité à approfondir la question [LRIM 88].

Il s'ensuit des choix précis que nous considérons comme indépendants des applications :

- à un instant donné, un menu est dit ***actif (visible)*** s'il est ***sélectionnable***, inactif sinon,

- la ***sélection*** d'un menu est toujours ***prioritaire*** sur celle d'un objet ou d'un caractère,

- à un instant donné, le système se trouve dans un certain ***état*** (contexte), la ***sélection*** d'un menu le fait passer dans un ***autre état***,

- un menu (et l'action associée) est lié à un ***concept général*** qui peut regrouper des sous-concepts qui affinent ce concept principal. Des menus différents peuvent être associés aux différents ***sous-concepts*** et sont donc rattachés au menu principal. Ceci définit une ***arborescence*** de sous-menus (une famille),

- en dehors des arborescences, le concepteur d'interface peut définir d'autres relations entre des menus liés à des concepts différents. Ces relations sont spécifiées selon les besoins, et fixent un ***lien de compatibilité*** entre deux concepts :

  + dans un contexte donné, la sélection d'un menu ***"incompatible"*** avec le concept en cours provoque une interruption brutale de l'action en cours et lance l'action associée au menu sélectionné. On appelle un tel menu ***Différé*** du contexte.

  + la sélection d'un menu ***"compatible"*** est interprétée comme la volonté de suspendre momentanément l'action en cours, pour exécuter l'action associée au menu sélectionné, et revenir au contexte de départ au point précis de l'interruption. On appelle un tel menu ***Immédiat*** du contexte.

  + un menu peut n'avoir de signification qu'à un instant précis d'une action, et ne pas exister à d'autres moments. Il permet de ***préciser le contexte*** en cours. S'il est sélectionné, le contexte est momentanément suspendu, il reprendra au point suivant l'interruption. On appelle un tel menu ***Local*** au contexte.

Remarque : Contrairement à un Local, un Immédiat (respectivement un Différé) existe indépendamment du menu dont il est Immédiat (respectivement Différé).

Afin de montrer le caractère général de ces notions, prenons un exemple dans un domaine autre que la C.A.O.

Soit une application qui offre les options suivantes :

- **FICHIER (concept principal)**

  Créer (Sous-menu)
  Détruire (Sous-menu)
  Renommer (Sous-menu)

- **EDITEUR** (concept principal)

  chercher *(local)*
  souligner *(local)*
  bloc *(local)*

- **AIDE** (concept principal)

Supposons que **FICHIER** soit un *DIFFERE* d'**EDITEUR**, et que **AIDE** soit un *IMMEDIAT* d'**EDITEUR**.

Au départ, seules les options principales apparaissent. Elles sont toutes dans une même couleur (rouge).

Supposons que l'on choisisse **EDITEUR**. Ce menu change de couleur pour indiquer qu'il devient le contexte en cours (jaune). Ses *locaux (chercher, souligner et bloc)* apparaissent dans une autre couleur (magenta) et deviennent actifs (sélectionnables).

A un instant donné, l'action attend un caractère à éditer.

Si l'on sélectionne **FICHIER** (resté en rouge), on sort définitivement de l'éditeur (car il y a ***incompatibilité***), des sous-menus apparaissent et on peut choisir l'un d'eux pour travailler avec le concept FICHIER.

Si l'on sélectionne AIDE (au lieu de FICHIER), qui a changé de couleur (devenu vert en entrant dans EDITEUR), on obtiendra des informations complémentaires sur l'utilisation de l'option en cours (EDITEUR). A la suite de quoi, on se retrouvera dans l'éditeur en attente d'un caractère. Bien qu' **AIDE** et **EDITEUR** soient des concepts différents, ils restent ***compatibles*** (Immédiat l'un de l'autre) et AIDE n'apporte rien sur la valeur du caractère attendu.

Si l'on sélectionne ***souligner*** (au lieu d'AIDE), le caractère attendu sera souligné. Ce choix a précisé (***local***ement) le contexte de l'**EDITEUR**.

On trouvera d'autres exemples dans [LRIM 88] et en annexe 1. Bien que cela semble paradoxal, il nous a paru impératif d'imposer certaines règles dont l'utilisateur doit être conscient à tout moment.

I.3.5. CONVENTIONS

En dehors des points très généraux abordés au paragraphe précédent, il en est d'autres qui nécessitent une attention particulière de la part de l'utilisateur. Il s'agit d'aspects plus typiques d'un système graphique.

A un instant donné, l'application peut demander un scalaire, une (ou plusieurs) coordonnée(s) ou une chaîne, ... Afin de décider des intentions, nous avons établi des conventions (pouvant être remises en question) d'***utilisation du clavier et de la souris.***

Rappelons que la souris possède deux boutons (gauche et droite):

- une pression sur le bouton de droite valide simultanément deux coordonnées X et Y ou X et Z ou Y et Z (sans interdire la sélection d'un menu ou d'un objet),

- une pression sur le bouton de gauche valide la 1ère coordonnée X ou Y ou Z suivant l'ordre établi par l'utilisateur (sans interdire la sélection d'un menu ou d'un objet),

- à tout moment, une pression sur la touche X (respectivement Y ou Z) établit l'ordre sur les coordonnées, X puis Y ou Z (respectivement Y puis X ou Z) et permet de revenir sur une coordonnée déjà validée,

- à tout moment, l'introduction d'une chaîne est interprétée avec les priorités suivantes :

  1) nom d'un menu à sélectionner
  2) nom d'un objet à identifier
  3) (par défaut) valeur de la 1ère coordonnée attendue
  4) (un message est associé) valeur d'un scalaire si la chaîne forme une expression grapho-numérique, la chaîne elle-même sinon.

- on peut momentanément changer les priorités en commençant la chaîne par un caractère spécial :

  '=' : signifie que la priorité est accordée à un scalaire (ou une chaîne) par rapport aux coordonnées,
  '^' : signifie que la priorité est accordée à une identification d'objet par rapport aux coordonnées,

- lors de l'introduction d'une chaîne (au sens large), on peut :

  + désigner graphiquement un objet dont le nom prendra automatiquement place dans l'expression.

  + sélectionner un menu dont l'action s'exécutera suivant les règles établies au paragraphe I.3.4. (Différé, Immédiat ou Local).

- lors d'une désignation d'objets, il peut y avoir plusieurs candidats. On les passe en revue en appuyant sur la "barre espace" et on confirme son choix par "retour chariot" ou "pression souris".

Toutes ces conventions peuvent sembler contraignantes. Notre expérience nous autorise à penser qu'il n'en est rien car:

- elles sont compréhensibles, donc faciles à apprendre,

- leur nombre reste limité, elles sont donc faciles à retenir,

- elles sont les mêmes à tout moment, cette constance renforce la consistance,

- elles sont associées à des mises en évidence visuelles (couleur, position, message, ...),

- les cas particuliers (changement de priorité) se présentent rarement. Ils répondent surtout à la volonté de pouvoir utiliser le clavier pour des intentions plus faciles (et plus naturelles) à spécifier à la souris (menus, objets, ...),

- la légère "frustration" que peut ressentir l'utilisateur nous paraît largement compensée par l'intérêt des horizons qui s'ouvrent à lui.

D'un point de vue plus formel, ces conventions correspondent à ce que certains auteurs appellent "techniques d'interaction" [FOL 82][SCH 85][KAM 83][GAL 83] et font l'objet de recherches spécifiques qui ont des incidences sur :

- l'indépendance par rapport au matériel,

- la consistance,

- la formalisation des outils offerts aux programmeurs d'applications et aux concepteurs d'interfaces.

Les points que nous venons de citer donnent une idée des problèmes que nous aurons à résoudre. Il ne nous semblait pas nécessaire de détailler davantage les fonctionnalités offertes par SACADO à un utilisateur final (Notice Utililisateur). Des exemples pratiques (et opérationnels) nous servirons de support pour des explications plus formelles des notions intuitives introduites dans cette partie.

Avant d'entrer dans le vif du sujet, il est important de situer l'interface homme-machine dans une architecture répondant à une démarche fondamentale.

DEUXIEME PARTIE

ARCHITECTURE LOGICIELLE

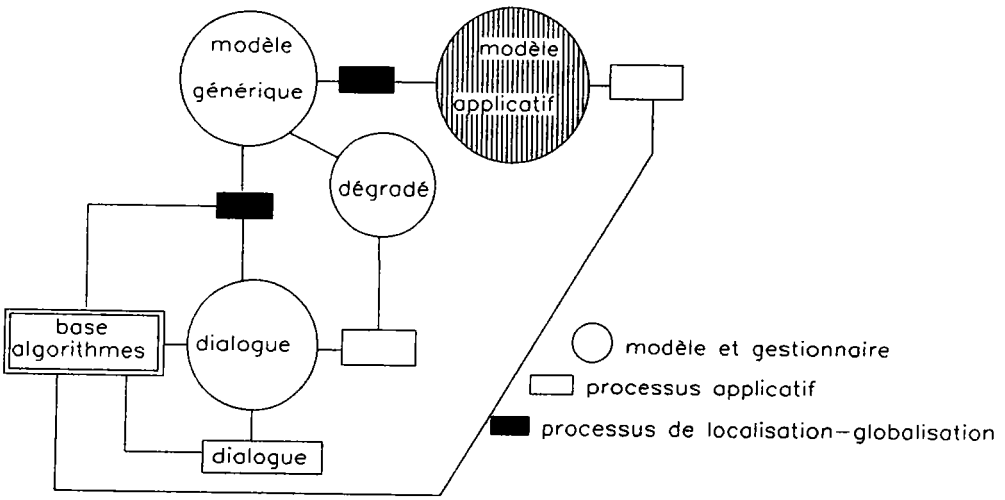
II.1. INTRODUCTION

Les problèmes posés en matière de modélisation des systèmes de C.A.O. font l'objet de recherches approfondies qui sortent du cadre de cet exposé. Le lecteur intéressé par ces questions trouvera une étude bibliographique dans [HEM 86]. Une réponse possible a vu le jour dans le système NADRAG [GAR 82][GAL 83]. L'auteur a affiné son approche indépendamment de ce système [GAR 83]. Ses travaux sur la méthodologie de développement, et notamment son étude très complète des difficultés à modéliser les systèmes graphiques interactifs, l'ont conduit à des propositions relatées dans [GAR 86] et [GAR 89b]. Celles-ci ont largement inspiré l'architecture de SACADO. Notre propos n'est pas d'entrer dans les détails, mais de rappeler les principes. Cela nous semble indispensable à plusieurs titres. Les notions que nous allons introduire sont partie intégrante de la philosophie de l'ensemble du système. Le modèle de dialogue doit être parfaitement situé avant d'en établir les fondements. Celui-ci est en relation avec d'autres modèles essentiels, tant par leur place dans l'ensemble (modularité, ouverture, optimisations), que par leur rôle vis-à-vis de l'interface homme-machine (convivialité).

Précisons que, dans cette partie, nous nous contentons de citer (et d'expliquer) les composants qui nous semblent significatifs du principe et qui sont nécessaires à la compréhension des parties à venir. De plus, nous nous limiterons à l'aspect 2D du système, dans la mesure où il a fait l'objet d'une implantation effectivement intégrée et complètement opérationnelle.

II.2. ORGANISATION DES DIFFERENTS MODELES

Pour un projet donné, la diversité des informations manipulées (objets géométriques et textuels, fonctionnements, assemblages, structurations, fabrications, ...) et la spécificité des traitements (visualisations, dialogues, calculs complexes, usinages, ...) ont conduit à un modèle central : le ***modèle générique*** [GAR 86][GAR 89a]. Celui-ci contient essentiellement des informations géométriques, structurelles et fonctionnelles, ainsi que les algorithmes nécessaires à leur manipulation et leur représentation. Le modèle générique peut être considéré comme le coeur du système. Des ***modèles applicatifs*** (et leurs opérateurs) en sont extraits par des processus de ***localisation***, et fournissent ainsi des ***vues externes*** adaptées à des tâches spécifiques. Des processus de ***globalisation*** permettent de passer de modèles applicatifs au modèle générique. Par définition, les modèles applicatifs sont fonctionnellement liés au modèle générique. En conséquence, des modifications de ce dernier sont répercutées sur les premiers. Il peut exister des modèles, issus du modèle générique par localisation, ne subissant pas d'éventuelles modifications : on les appelle ***modèles dégradés*** (nous n'en parlerons pas davantage). La figure II.1. (tirée de [GAR 89a]) schématise ces notions.



(figure II.1.)

Différents projets ainsi définis peuvent être eux-mêmes considérés comme des extractions (par localisation) d'un modèle générique des activités de l'entreprise. Le principe général reste donc valable à différents niveaux d'abstraction.

II.3. LE MODELE GENERIQUE

Concernant le système SACADO 2D, le modèle générique est principalement composé de données géométriques (points, segments, cercles, arcs, ...), d'informations structurelles (niveaux de visualisation, contours, ...), de catalogues (pièces paramétées, ...), d'habillages (hachurages, cotations, textes, ...), d'attributs (couleur, nature, nom, ...) et d'algorithmes divers (accès, modifications, suppressions, transformations géométriques, changement de niveau de visu., ...).

Tout élément d'une scène (objet géométrique, habillage, ...) est modélisé par l'instanciation (valorisation) d'un schéma (structure logique) unique. La figure II.2. donne le schéma d'un objet de SACADO 2D.

Nature	**Nom**	**N° Visu.**
N° Pièce	**N° Contour**	**N° Cote**
Mode	**Couleur**	**Texture**

(figure II.2.)

Un processus de localisation consiste à extraire (ou transformer) tout ou partie des valeurs du modèle. Un ensemble de primitives (algorithmes) assure la transparence et la cohérence des diverses manipulations (localisations et globalisations). Ces procédures s'appuient principalement sur la nature des objets manipulés. Voici

quelques exemples de localisations et de globalisations définies dans SACADO 2D (les principes seront utilisés dans les exemples de la troisième partie) :

- C3PTS(p1,p2,p3,nc) qui, à partir de trois points (trois couples de coordonnées) crée (instancie) un cercle passant par ces trois points et dont le nom sera nc (il s'agit d'une globalisation),

- CCRAY(p,r,nc) qui, à partir d'un point et d'un scalaire, crée un cercle de centre p, de rayon r et dont le nom sera nc (il s'agit également d'une globalisation),

- MODIFEXTR1(ns,p) qui remplacera la première extrémité du segment nommé ns par le point p (cette modification sera répercutée, par localisation, sur les modèles concernés),

- RAYON(nc,r) qui initialisera le scalaire r à la valeur du rayon du cercle nc (c'est une localisation sans modifications).

On trouvera l'étude détaillée de ce modèle dans [MIN 88] et [JUN 89]. Outre des précisions concernant les structures et algorithmes développés dans ce cadre, ces notes fournissent des éléments importants liés aux limites d'un environnement microordinateur, ainsi que tout ce qui se rapporte au modèle 3D dont le modèle 2D présenté dans ce paragraphe n'est en fait qu'une extraction. Pour notre part, nous nous en tiendrons aux points ci-dessus, dans la mesure où ils suffisent amplement à la compréhension du sujet qui nous préoccupe ici.

II.4. LE MODELE DE VISUALISATION

Ce modèle est considéré comme un modèle applicatif dans la mesure où il est en partie composé d'informations extraites du modèle générique. Ce qui le caractérise davantage dans ce sens, c'est sa vocation à permettre l'optimisation de fonctions spécifiques. En fait, le terme "visualisation" est à prendre au sens large. En effet, la perception que l'on en a est surtout "visuelle", car ce modèle permet l'affichage : une image graphique n'est qu'une vue externe (au sens Bases de Données) du modèle générique (c'est d'ailleurs l'un des aspects les plus spectaculaires des systèmes graphiques modernes). Néanmoins, comme nous le verrons par la suite, ce modèle joue un rôle essentiel dans le dialogue (mais cela se "voit" moins). Il contribue fortement à accroître la convivialité du système. A ce titre, il convient de l'étudier avec soin. Il est aujourd'hui considéré comme indispensable par son rôle charnière entre le modèle générique et le modèle de dialogue.

Dans ce paragraphe, nous présentons le modèle de visualisation dans son utilisation pour l'affichage. Nous voulons surtout donner les grandes lignes qui ont conduit à son implantation dans SACADO.

II.4.1. LE PROBLEME ELEMENTAIRE

Les objets représentés dans le modèle générique sont définis dans un système de coordonnées réelles (donc continu), alors que les supports matériels (écrans, traceurs, imprimantes, ...) nécessaires à leur visualisation sont, par essence, discrets (coordonnées entières). La fonction première du modèle de visualisation est d'assurer le lien et la cohérence entre un espace utilisateur (modèle générique) et un espace écran (ou autre), et ce, indépendamment d'une quelconque unité (m, cm, mm, u, ...). Le souci dominant étant d'optimiser les différents traitements inhérents à ce processus.

Nous préconisons un modèle qui reste le plus proche possible du monde réel, et qui contient de façon quasi permanente le maximum d'informations permettant le passage d'un espace utilisateur (une fenêtre) à un espace écran (une clôture), et vice versa. A cet effet, nous conservons :

- la définition de la fenêtre (ses bords réels, de façon absolue),
- la définition de la clôture (ses bords relativement à un écran normé [réels] et de façon absolue [entiers]),
- les paramètres des transformations fenêtre-clôture (localisation) et clôture-fenêtre (globalisation),
- la caractérisation de la précision (epsilons, décimales significatives, ...) relativement à la machine, relativement à la fenêtre, relativement au couple (fenêtre,clôture).

Ces informations ne sont pas recalculées lorsqu'elles sont utilisées dans les algorithmes qui complètent le modèle de visualisation (centrage d'une scène, affichages à partir de données réelles, découpage en réels, récupération en réel d'un objet montré à l'écran, ...). Il s'agit d'accès directs à des structures qui peuvent être implantées de diverses façons (plus ou moins dynamiques).

Il faut remarquer que nos choix impliquent qu'aucune information de bas niveau relative à la représentation graphique (pixels) des objets du modèle générique n'est conservée. Notre approche se différencie fondamentalement de celles de type GKS [END 84] ou PHIGS [KRZ 86] qui ne conservent que des informations graphiques, sans lien avec le monde réel.

Parmi les intérêts que présente notre approche, les principaux sont :

- assurer la cohérence entre le modèle générique et son image à l'écran,

- permettre les répercussions immédiates des modifications,

- éviter la gestion coûteuse de mémoires images (gain de place et de temps),

- faciliter des opérations telles que le zoom (par empilement des fenêtres [et des coefficients de passage] intermédiaires),

- éviter des affichages (donc des transformations) inutiles,

- visualiser en permanence la trace réelle et significative de la souris se déplaçant sur l'écran,

- gérer efficacement les parties indéfinies (en dehors de la fenêtre réelle), tant au niveau du modèle qu'au niveau du dialogue,

- définir simplement des grilles réelles, pour permettre des constructions précises en utilisant la souris qui est forcément approximative,

- accroître la portabilité des programmes (sources d'applications) en ne modifiant que des primitives de bas niveau utilisées dans les algorithmes du modèle de visualisation (bibliothèques graphiques ou relatives aux organes d'entrées, donc proches des matériels).

II.4.2. GENERALISATION

Dans cet esprit, il nous a semblé intéressant (voire primordial) d'étendre ces notions à l'utilisation de plusieurs fenêtres et plusieurs clôtures [TOT 88]. Cela nous permettra :

- d'améliorer la convivialité et d'accroître les possibilités du système,

- d'optimiser davantage certaines fonctions spécifiques (affichage et dialogue) afin de diminuer les temps de réponses.

De façon schématique, il s'agit d'être capable de gérer les problèmes sous-jacents aux points cités dans le paragraphe précédent, tout en tenant compte de la multitude : définir N fenêtres, M clôtures, et autoriser toutes les combinaisons (associations) possibles.

C'est ce qui est communément appelé "multi-fenêtrage". De nombreux systèmes offrent, de façon externe, un tel environnement (Macintosh[MAC 88] , MS/Windows [MSW 88], X-Window [GET 87]). Néanmoins, il semble que la plupart de ces systèmes ne soient pas adaptés à la CFAO. En effet, ils ne gèrent que des informations

purement graphiques (au niveau du pixel ou de l'octet [mémoires images]) sans relation avec un espace utilisateur réel. Cela conduirait à des quantités d'informations prohibitives compte tenu de la complexité des scènes (nécessitant de nombreuses couleurs) que l'on peut rencontrer, d'une part, et des limites qu'impose un environnement microordinateur (dans l'état actuel de la technique en tout cas), d'autre part. De plus, et c'est ce qui nous paraît le plus important, notre but est de visualiser et de manipuler interactivement plusieurs vues, simultanément présentes à l'écran, de tout ou partie du modèle générique. Or, les systèmes précités ne permettent que la manipulation de plusieurs entités (ou versions d'une même entité) différentes, donc fonctionnellement indépendantes. Cette limite est un handicap sérieux (par exemple, une modification sur une vue n'est (et ne peut pas être) répercutée sur une autre). En fait, ces systèmes ne font que simuler plusieurs écrans (clôtures) sur un seul.

C'est pourquoi, il nous a semblé indispensable d'intégrer un modèle qui réponde à des besoins plus spécifiques. Le modèle présenté au paragraphe précédent est étendu de la façon suivante :

- définitions de toutes les fenêtres (positions, références aux objets qu'elles contiennent, priorités),

- définitions de toutes les clôtures (positions, priorités, interférences),

- associations des espaces (transformations, précisions),

- états (désignables, visibles, invisibles [relatifs et absolus]) des entités fenêtres, clôtures et niveaux de visualisation.

Ces choix se justifient car :

- il s'agit d'une généralisation systématique du cas trivial (dans la mesure où, à un instant donné, on ne travaille que par rapport à un unique couple (fenêtre,clôture) courant [car il n'y a qu'une seule souris], même si cela a des incidences sur les autres vues),

- cela ne nécessite pas un grand espace mémoire par rapport à l'utilisation de mémoires images dont le nombre serait accru par la multitude des espaces représentés,
- dans la mesure où l'environnement est spécifié, la majeure partie des informations est statique, donc calculable et stockable de façon quasi définitive. L'accès direct semble donc plus indiqué que le calcul des informations à chaque fois que cela est nécessaire,

- l'association des fenêtres à leur contenu (des objets) permet d'optimiser l'affichage et l'identification graphique réels (c'est un filtre qui restreint ponctuellement le nombre d'objets à manipuler),

- les informations de priorité et d'interférence des clôtures (relativement à d'autres clôtures) permettent d'optimiser l'affichage (multi-découpage) et l'identification graphique de bas niveau, ainsi que l'obtention de la trace réelle de la souris,

- les priorités relatives des fenêtres autorisent une gestion efficace de n fenêtres associées à une seule clôture,

- les états (relatifs et absolus) des fenêtres, clôtures et niveaux de visualisation sont des facteurs importants d'amélioration de la convivialité.

Dans le modèle de visualisation, il est fait référence à des objets du modèle générique. En ce sens, ces deux modèles sont fonctionnellement dépendants l'un de l'autre. En revanche, leurs implantations respectives demeurent indépendantes l'une de l'autre (exceptés certains algorithmes de localisation et de globalisation). Il en va de même en ce qui concerne les relations qui peuvent exister entre le modèle de dialogue et le modèle de visualisation. Ces dernières remarques semblent évidentes en raison des définitions même du modèle générique et des modèles applicatifs. En fait, par la suite, nous nous apercevrons que le modèle de dialogue est un modèle générique de l'architecture logicielle. En ce sens, il est encore plus indépendant des domaines d'applications. Notre conviction est que le modèle générique, le modèle de visualisation et le modèle de dialogue constituent un ensemble impérativement intégré : à la limite, cet ensemble doit être considéré comme un modèle générique (pratiquement indivisible) de l'interface d'un système graphique interactif.

L'annexe 2 fournit des figures qui schématisent une implantation possible (celle de SACADO) des structures du modèle de visualisation. Il ne nous paraît pas utile de les détailler davantage ici.

II.5. LE MODELE D'INTERFACE : PREMIERE VERSION

Le modèle de dialogue occupe une place prédominante (centrale) dans l'architecture logicielle. C'est pourquoi, dans ce paragraphe, nous en donnons une version simplifiée indépendamment du modèle générique (cf. II.2. et II.3.). Nous définissons les structures logiques et les mécanismes à mettre en oeuvre pour la réalisation du MONITEUR DE DIALOGUE (cf. figures I.1. et I.2.). Celui-ci s'appuie essentiellement sur:

- la définition et la gestion des menus,
- l'interprétation des intentions,
- l'exécution des actions associées.

Il faut donc caractériser chacun de ses composants afin de spécifier leurs places et rôles dans le modèle. Tous répondent à des exigences particulières.

REMARQUE : Nous précisons (et cela vaut pour tout le reste de l'exposé) que les figures présentées sont à prendre comme des schémas fonctionnels des structures logiques introduites. Sur ces figures, les flèches indiquent soit des listes (au sens large) d'entités, soit des références à des entités.

II.5.1. STRUCTURE DES MENUS

Un concepteur doit pouvoir développer une interface indépendamment du code de l'application, afin de :

- séparer les rôles (concepteur et programmeur),
- réaliser rapidement un prototype de simulation qui sera affiné par touches successives,
- pouvoir modifier l'interface sans toucher au code application et sans le recompiler,
- permettre d'adapter l'interface à un utilisateur particulier (gaucher ou droitier, ...).

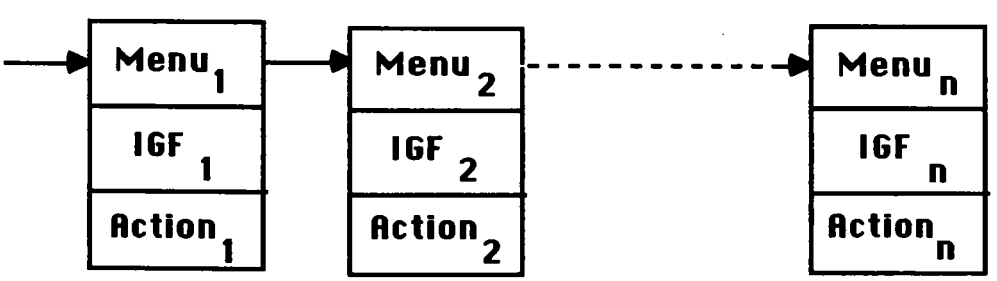
En conséquence, le modèle doit garantir le dynamisme :

- de l'aspect externe des menus (visibilité, position, couleur, forme, ...),
- de l'association menu-action (attachement et déclenchement),
- des notions de compatibilité et de localité (aspect, interrelations, ...).

Par ailleurs, le programmeur d'application ne doit pas se soucier de ces problèmes. En d'autres termes, aucun de ces points ne doit être pris en compte dans le code. Les intérêts sont :

- assurer l'indépendance dialogue-code,
- simplifier la tâche du programmeur,
- faciliter la maintenance,
- garantir la consistance.

Les informations sont stockées dans une structure (dynamique) qui constitue une partie de la base de données du MONITEUR, au lieu d'une spécification procédurale liée à l'application. Soit, en première approche, une liste des menus existants :



(figure II.3.)

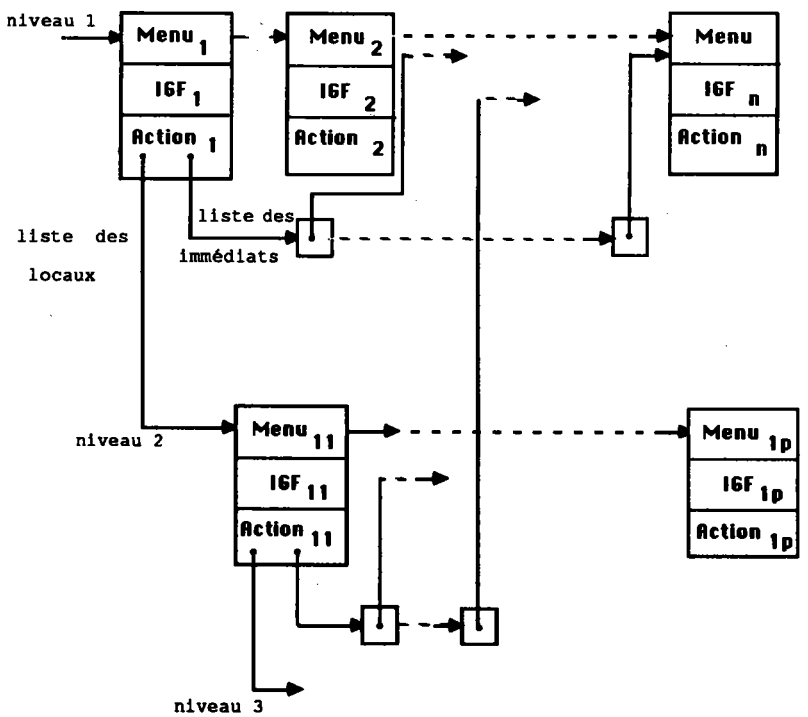
Où, pour tout i :

- $Menu_i$ est un identificateur (numéro, nom, ...),

- IGF_i est l'ensemble des informations graphiques et fonctionnelles (visibilité, position, couleur, forme, ...) concernant $Menu_i$,

- $Action_i$ est un identificateur externe du traitement (code) interne à exécuter si $Menu_i$ est sélectionné.

Les notions de compatibilité et de localité nécessitent d'affiner le schéma (cf. figure II.3.). Ces propriétés sont liées à l'action associée puisqu'elles interviennent lors de son exécution.



(figure II.4.)

Remarques :

- la structure représente tous les menus de l'application, qu'ils soient actifs ou non,

- une action est dite terminale si elle ne sollicite pas l'utilisateur,

- l'action associée à un menu possédant des sous-menus se limite à l'activation de ces mêmes sous-menus. Cette action n'étant pas interactive, elle ne peut pas posséder des locaux,

- excepté dans le cas des sous-menus, les niveaux n'indiquent pas une arborescence. Il s'agit de niveaux de coexistence (coactivité). Les menus d'un même niveau sont susceptibles d'être actifs en même temps,

- le niveau 1 représente l'ensemble des menus M_1, ..., M_n actifs au début, les seuls que l'utilisateur voit (et peut sélectionner) quand il commence à travailler,

- si des locaux d'un niveau i sont actifs, alors des locaux des niveaux 1 à (i-1) le sont également, Plus précisément, soient M_i et M_j au niveau 1 :

  + M_{i1}, ..., M_{ip} les locaux de Mi sont au niveau 2,
  + M_{j1}, ..., M_{jk} les locaux de M_j sont également au niveau 2,
  + M_{im1}, ..., M_{imt} les locaux de M_{im} sont au niveau 3,

  Si M_{im1}, ..., M_{imt} sont actifs alors :

  + M_{i1}, ..., M_{ip} sont actifs,
  + M_1, ..., M_n sont actifs,
  + M_{j1}, ..., M_{jk} ne sont pas forcément actifs. Cela dépend du contexte et des liens de compatibilité entre M_i et M_j,

- si l'on pose Ni l'ensemble des menus de niveau i, alors pour tout i différent de j,

$$N_i \cap N_j = \varnothing$$

  autrement dit, les locaux sont entièrement et effectivement définis comme menus,

- les Immédiats et Différés sont exclusifs les uns des autres (un menu ne peut pas être à la fois compatible et incompatible avec un autre), ce qui explique qu'on ne conserve que les Immédiats (les Différés pouvant en être déduits),

- les éventuels Immédiats d'un menu situé au niveau i sont déjà définis aux niveaux 1 à i , la liste des Immédiats n'est donc pas une duplication des menus impliqués, c'est une référence effective à des définitions uniques.

Nous nous attacherons à mieux définir ces points lorsque nous aurons encore plus affiné le modèle.

Nous venons de schématiser fonctionnellement la structure de menus. Continuons dans le même état d'esprit avec l'exécution d'une action.

II.5.2. DECLENCHEMENT PROCEDURAL

Une action est attachée à un menu au moyen d'un identificateur externe faisant partie de la définition même du menu. Nous allons décrire une version simplifiée des mécanismes qui en permettent l'exécution. Un de nos objectifs est de décharger le programmeur de la gestion du dialogue (séparation code-dialogue), c'est pourquoi nous avons choisi d'interpréter le lancement d'une action.

Le schéma d'une application interactive consiste à boucler sur la possibilité de sélectionner une option jusqu'à décider de quitter l'application.

```
APPLICATION

 BOUCLER
     sélectionner une option ;
     CHOISIR  option  PARMI
             opt1 : fct1 ;
             opt2 : fct2 ;
              .
              .
             optn : fctn ;
     FIN CHOIX
  JUSQU''A  option = quitter

FIN APPLICATION
```

(figure II.5.)

La figure II.5. montre un modèle classique [PAR 69] où le contrôle du flot d'entrées apparaît explicitement dans le code de l'application. Les inconvénients sont nombreux :

- le programmeur et le concepteur ne font qu'une seule et même personne (non séparation des rôles),
- la boucle doit être réécrite pour chaque application (elle en est dépendante),
- toute modification d'options nécessite une recompilation (ralentit le développement),
- l'association menu-action est statique,
- le principe conduit à des chemins uniquemement arborescents (Immédiats, Différés et Locaux difficiles à prendre en compte).

Néanmoins, le fait de boucler sur une sélection d'option est un principe qui reste valable pour la quasi-totalité des applications. On peut donc l'écrire une fois pour toute et l'intégrer dans le modèle d'une interface "standard". Nous proposons donc un noyau qui se charge de déclencher indirectement (implicitement) les actions. Pour ce faire, nous avons défini une primitive d'appel qui, à partir d'une option (donc de son action) exécute le bon traitement.

```
APPEL (option)
  CHOISIR  option.action PARMI
          act1 : fct1 ;
          act2 : fct2 ;
           .
           .
           .
         actn : fctn ;
  FIN CHOIX
FIN APPEL
```

(figure II.6.)

où acti est l'identificateur externe de l'action associée à opti et fcti le code effectif.

Cette primitive est utilisée dans le noyau "standard" qui réalise la boucle sur les entrées.

```
NOYAU

BOUCLER
   sélectionner une option ;
   APPEL (option) ;
JUSQU'A  option = quitter

FIN NOYAU
```

(figure II.7.)

Nous aurons l'occasion d'y revenir, mais d'ores et déjà faisons quelques remarques :

- le principe du NOYAU est général. Ni le concepteur, ni le programmeur n'ont à s'en soucier,

- le programmeur n'écrit que le code des fonctions,

- le concepteur définit l'interface en initialisant la structure de menus,

- le lancement explicite d'une fonction n'apparaît que dans "APPEL", qui fait le lien entre noms externes et internes. En dehors du code proprement dit des fonctions, c'est le seul module à recompiler dans le cas d'une modification profonde de l'application. Nous verrons comment le processus peut être assoupli en en automatisant une partie.

Tous ces points peuvent être rendus transparents. Dès lors, si l'on fournit des outils appropriés au concepteur et au programmeur, on pourra simplifier leurs tâches tout en garantissant un résultat qui réponde efficacement à tous les problèmes déjà soulevés.

Donnons des exemples du dynamisme qu'apporte cette solution. Supposons qu'une interversion ait été commise dans des appels de fonctions (exécuter fct_j pour opt_i et fct_i pour opt_j). Dans une approche classique, nous aurions :

```
BOUCLER
  sélectionner une option ;
  CHOISIR option PARMI
        .
        .
      opti : fctj ;
     optj : fcti ;
        .
        .
  FIN CHOIX
JUSQU'A option = quitter
```

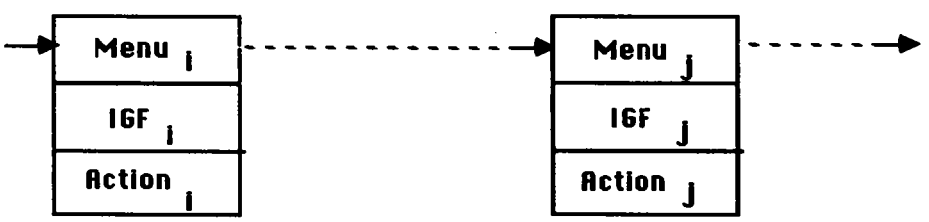
à modifier en :

```
BOUCLER
  sélectionner une option ;
  CHOISIR option PARMI
        .
        .
      opti : fcti ;
     optj : fctj ;
        .
        .
  FIN CHOIX
JUSQU'A option = quitter
```

et à recompiler.

(figure II.8.)

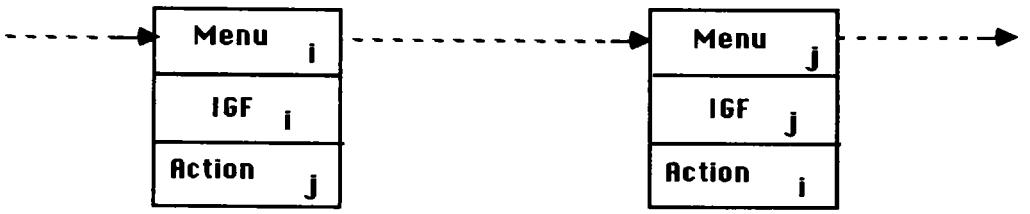
Alors qu'avec notre modèle, nous aurions :




```
+ LE  NOYAU

+
   APPEL (option)
     CHOISIR  option.action PARMI
            act1 : fct1 ;
            acti : fctj ;
              .
              .
           actj : fcti ;
     FIN CHOIX
   FIN APPEL
```

(figure II.9.)

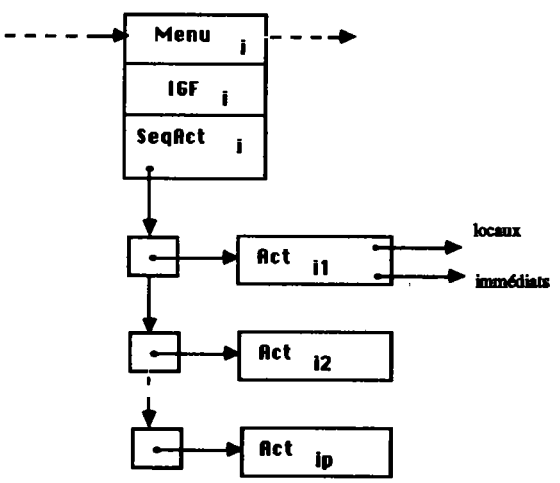
à modifier en :



(figure II.10.)

sans aucune recompilation, en intervertissant $Action_i$ et $Action_j$ dans le modèle, plutôt que dans le code.

Cette première version donnait le principe de la relation entre le modèle (structure des menus) et le code (APPEL). Nous donnons une première extension qui facilitera la définition et la manipulation de cas plus complexes que ceux définis par les figures II.4. et II.6. . En fait, nous associons à un menu une séquence d'actions (qui peut se limiter à une seule). Un menu est donc défini par:



(figure II.11.)

De plus, la primitive "APPEL" prend cette extension en compte :

```
APPEL (option)
   BOUCLER  sur option.SeqAct
      CHOISIR  option.SeqAct.action PARMI
             act1 : fct1 ;
             act2 : fct2 ;
              .
              .
              .
            actn : fctn ;
      FIN CHOIX
   JUSQU'A  Fin option.SeqAct
FIN APPEL
```

(figure II.12.)

Cela augmente le dynamisme de l'application. Notamment, il arrive souvent que l'on désire ajouter une macro-option comme étant la séquence d'options déjà existantes. Par exemple, opti = optj; optk; optm qui exécuterait directement fctj, fctk et fctm en séquence, en ne sélectionnant que opti. Le schéma classique donnerait :

```
BOUCLER
  sélectionner une option ;
  CHOISIR option PARMI
        .
      optj : fctj ;
      optk : fctk ;
      optm : fctm ;
        .
    FIN CHOIX
JUSQU'A option = quitter
```

(figure II.13.)

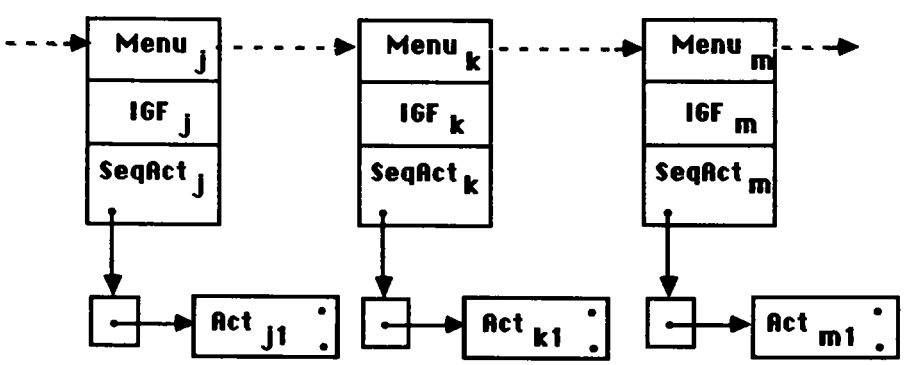
à modifier en :

```
BOUCLER
  sélectionner une option ;
  CHOISIR option PARMI
        .
      optj : fctj ;
      optk : fctk ;
      optm : fctm ;
        .
      opti : fctj; fctk; fctm;
    FIN CHOIX
JUSQU'A option = quitter
```

(figure II.14.)

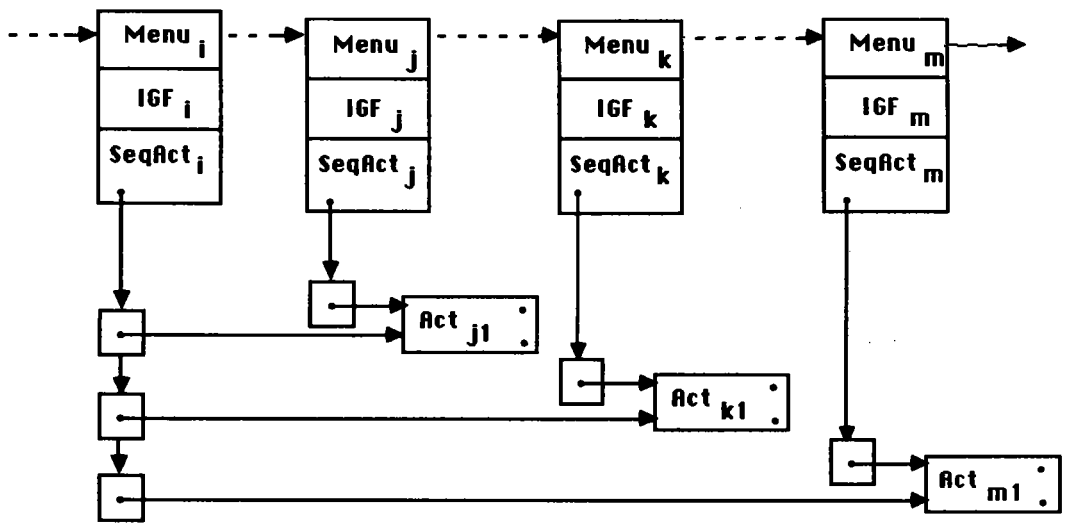
et à recompiler.

Alors qu'avec notre modèle, cela devient :



(figure II.15.)

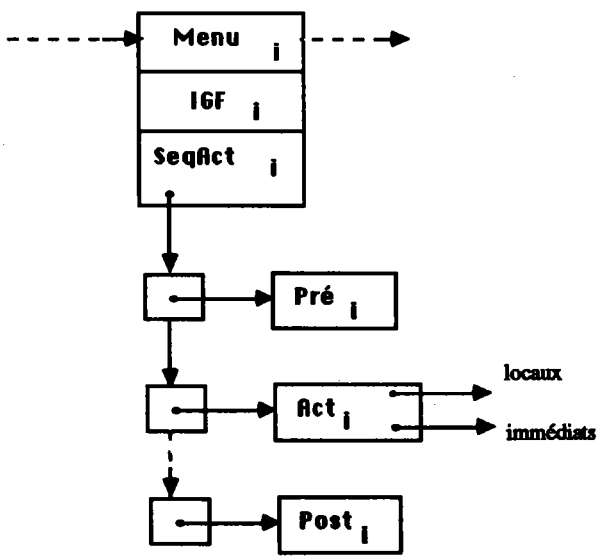
à modifier en :



(figur II.16.)

sans modification, ni recompilation d'"APPEL" ou de l'application, en définissant (dans le modèle) un menu $Menu_i$ dont la séquence d'actions fait référence à des actions (Act_{i1}, Act_{k1}, Act_{m1}) déjà existantes dans le modèle.

Il faut remarquer que opt_i hérite automatiquement de toutes les propriétés de ses composants. D'autre part, il est à noter que cette structure permet de prévoir les pré-traitements et post-traitements, qui s'intègrent bien dans le modèle (figure II.17.).



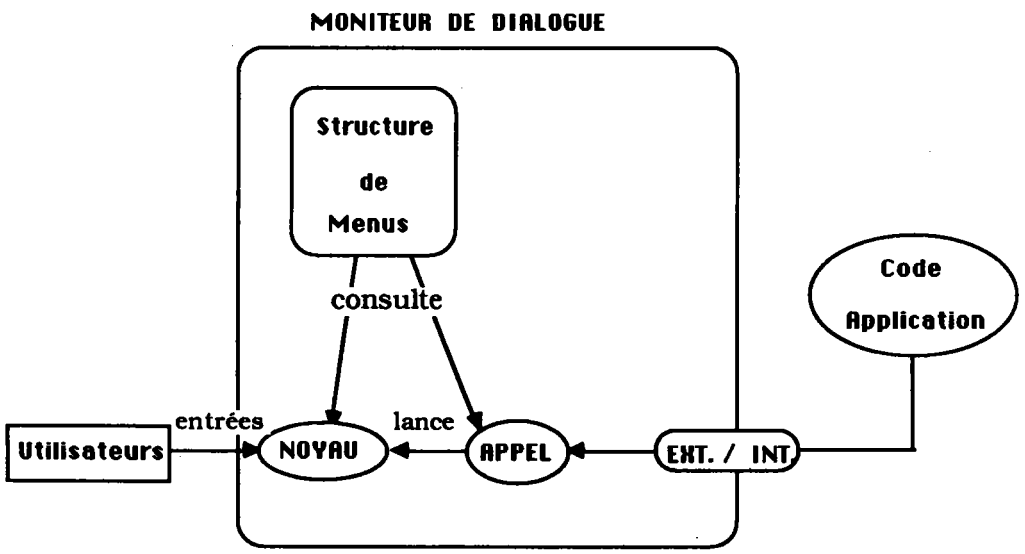
(figure II.17.)

Nous verrons également comment cette approche permet de définir facilement des traitements récursifs autorisant des dialogues complexes et augmentant davantage la puissance de description du modèle.

Le paragraphe suivant donne un schéma récapitulatif de cette première version.

II.5.3. RESUME

Il nous paraît important de resituer nos propos par rapport au schéma d'ensemble.



(figure II.18.)

Les flèches indiquent le sens de circulation des informations en accord avec cette première modélisation. L'utilisateur donne une entrée à partir de laquelle le noyau détermine une option provenant de la structure de menus. Le noyau lance "APPEL" qui retrouve et exécute l'action associée en fonction de la correspondance nom externe-nom interne.

Conformément à ce qui a été vu jusqu'à présent, la figure ne considère qu'un type d'entrée particulier, à savoir, la sélection d'une option. Nous ne parlons pas encore du problème des sorties que nous définirons ultérieurement dans une partie consacrée à la caractérisation d'une action. De même, nous ne faisons pas apparaître qui gère l'aspect visuel des menus (visibilité, couleur, ...).

Pour bien situer ces points, il est nécessaire d'aller plus loin dans le raffinement du modèle.

TROISIEME PARTIE

LE MODELE COMPLET DE L'INTERFACE

III.1. AMELIORATION DU MODELE

La première version du modèle ne précisait pas les solutions adoptées quant à l'interprétation d'une intention autre que le choix d'une option au niveau le plus haut de la structure de menus. Il nous faut donc avancer dans le dialogue entre l'utilisateur et l'application pour apporter des éléments de réponse. Nous allons voir comment sont gérées les notions de compatibilité et de localité.

Nous serons amenés à introduire de nouveaux concepts fondamentaux et caractéristiques de notre approche. Nous aboutirons ainsi à un modèle enrichi dans ses structures et mécanismes. Ce n'est qu'alors qu'il nous semblera suffisamment significatif pour poser les problèmes rencontrés, et suffisamment représentatif de nos idées pour le situer par rapport à d'autres modèles.

III.1.1. CARACTERISATION D'UNE INTENTION

Un point essentiel de notre approche est de permettre un maximum d'interventions à l'utilisateur. C'est volontairement que nous avons employé jusqu'ici le terme d'intention dans son sens le plus général possible. Ceci traduit notre volonté de défendre un point de vue qui différencie fondamentalement cette approche de ceux qui considèrent plusieurs types d'entrées [FOL 82] [ROS 82]. A savoir :

- la sélection pour le choix d'une option,
- la désignation pour le choix d'un objet,
- la localisation pour la récupération de deux coordonnées (X et Y),
- la valuation pour la récupération d'un scalaire,
- le texte pour la récupération d'une chaîne de caractères.

Cette distinction peut être acceptée fonctionnellement dans la mesure où elle peut être à la base de l'interprétation d'une intention. Par contre, elle nous semble inadaptée si elle demeure effective dans le modèle, c'est-à-dire si elle conduit à des primitives de dialogue distinctes (une par type d'entrée).

Du point de vue de l'utilisateur final, il s'agit d'une limite aux possibilités offertes, puisqu'à un instant donné, il ne peut rien faire d'autre que ce qui est prévu par la primitive qui le sollicite. Par exemple, il lui est impossible de sélectionner un menu quand l'application attend un scalaire. Le contexte dans lequel il se trouve est figé, ce qui va à l'encontre de notre souci de dynamisme.

Du point de vue du programmeur, cela nécessite la connaissance et la compréhension de chaque primitive, afin de les utiliser à bon escient. En fait, nous pensons que de la même façon qu'il ne faut pas noyer l'utilisateur dans le matériel, il ne faut pas noyer le programmeur dans le logiciel. Il nous semble que cette variété ne fera qu'accroître la difficulté de caractériser un traitement

interactif et, par là même, compliquera les outils mis à la disposition du programmeur.

Cette approche est étroitement liée au contrôle entièrement interne du flot d'entrées, pour lequel elle se justifie aisément, car dans ce cas, tout apparaît explicitement dans le code.

Malheureusement, elle ne permet pas une prise en compte satisfaisante des notions de compatibilité et de local. Il est impératif de les intégrer au concept d'interaction, et ce de la façon la plus transparente possible, puisqu'il est hors de question de laisser leur gestion à la charge du programmeur.

En conséquence, nous proposons une primitive de dialogue unique que nous appelons ***"INTERACTION"*** et qui a pour rôle :

- d'interpréter une intervention de l'utilisateur suivant les conventions que nous avons fixées,
- de restituer une série de résultats qu'un traitement peut utiliser (objets, coord., scalaire, état du système, ...),
- de gérer les Immédiats, Différés et Locaux.

Soit une version simplifiée :

```
INTERACTION
  DEBUT
    Différé<--faux ;
    BOUCLER
      SI Différé=faux
      ALORS
            Interpréter l'intention ;
      FIN SI

      SI un menu
      ALORS
            SI pas un DIFFERE
            ALORS
                  APPEL(menu) ;
            SINON
                  Différé<--vrai ;
            FIN SI
      FIN SI

  JUSQU'A Validation

FIN INTERACTION
```

(figure III.1.)

La primitive boucle afin de gérer les Immédiats. En effet, si le menu appelé est Immédiat, son action doit s'exécuter et l'interaction doit reprendre comme si rien ne s'était produit (retour au point précis de l'interruption).

La condition d'arrêt (validation) est à prendre au sens large. Elle dépend du résultat de l'interprétation de l'intention (objet, scalaire, abandon, nature d'un menu, ...).

Noter qu'un Différé a été sélectionné signifie que les résultats de l'interaction ne seront pas utilisés. Le traitement appelant "INTERACTION" en sera averti afin de réagir en conséquence.

L'appel d'un menu conduira éventuellement à des imbrications d'"INTERACTION". Notamment, cela pourra correspondre à une descente dans les niveaux de locaux. Si, à un instant donné, on se trouve dans l'exécution d'un menu de niveau i et que l'on sélectionne un Différé de niveau j (1 <= j <= i), l'information devra être automatiquement remontée jusqu'au niveau j, où le menu pourra être appelé. C'est le rôle du booléen global "différé", qui assure la transmission d'un niveau à un autre. Il va sans dire que le menu sélectionné est également transmis. Ainsi, il s'exécutera automatiquement au niveau j, sans que l'utilisateur ait besoin de le sélectionner à nouveau.

Si l'on considère les menus du niveau 1 comme les locaux d'un traitement particulier qui n'est autre que le ***NOYAU*** que nous avons défini au paragraphe II.5.2. , on s'aperçoit que dans le ***NOYAU*** , la sélection d'une option suivie de son ***APPEL*** n'est autre qu'un cas particulier d'***"INTERACTION"*** . On peut donc généraliser et compléter le NOYAU :

```
NOYAU
Initialisations ;
BOUCLER
   INTERACTION;
JUSQU'A quitter
Terminaisons ;
FIN NOYAU
```

(figure III.2.)

La boucle reste nécessaire car au niveau 1, un menu sélectionné est un Local de l'action "NOYAU", et non un Immédiat. On sort donc d'"INTERACTION" après son appel. Par contre, au niveau 1, la gestion des Différés reste de la responsabilité d'"INTERACTION".

Comme nous l'avons vu, les Locaux d'une action n'ont de sens que lors de son exécution. En conséquence, ils sont activés avant et désactivés juste après. De même, les Immédiats changent d'aspect pour avertir l'utilisateur. Il est naturel de confier ce travail à la primitive "APPEL". De plus, c'est elle qui transmet un éventuel Différé. D'où une nouvelle version :

```
APPEL (option)

   BOUCLER  sur option.SeqAct
      Activer Locaux ;
      Changer aspect Immédiats;
      CHOISIR  option.SeqAct.action PARMI
            act1 : fct1 ;
            act2 : fct2 ;
             .
             .
           actn : fctn ;
   FIN CHOIX
      Rechanger aspect Immédiats;
   Désactiver Locaux ;
   JUSQU'A  Fin option.SeqAct ou Différé
   SI Différé
   ALORS  option<-- menu DIFFERE ;
   FIN SI

FIN APPEL
```

(figure III.3.)

III.1.2. CARACTERISATION D'UNE ACTION INTERACTIVE

Ce paragraphe montre la forme du code d'une action obtenu par application des principes énoncés ci-dessus (sans décrire le mode d'obtention).

Néanmoins, animés par notre souci de généralité, nous essaierons de fournir une définition qui facilitera l'écriture du code tout en profitant de la puissance du modèle. Ce qui nous donnera l'occasion de faire apparaître les problèmes à résoudre pour satisfaire le programmeur.

En première approche, on peut considérer qu'il y a deux types d'actions :

- les action terminales (AT) qui n'effectuent que des calculs (au sens large) et qui ne sollicitent pas l'utilisateur,
- les actions interactives (AI) qui mêlent calculs et interventions de l'utilisateur.

Dans notre modèle actuel, ces deux formes d'actions n'ont ni paramètres d'entrée, ni paramètres de sortie. Elles travaillent avec des variables locales. Nous verrons par la suite dans quelle mesure cela peut être gênant. Notre expérience nous permet de dire que l'approche reste acceptable. En fait, les actions sont associées à des fonctionnalités (concepts) différentes les unes des autres. Elles n'ont donc, a priori, aucune raison de se communiquer des informations. D'où les expressions générales:

```
ACTION TERMINALE
VL : variables locales ;
 Calcul(VL) ;
FIN AT
```

(figure III.4.)

où **Calcul** est une fonction qui manipule en entrée et/ou sortie les Variables Locales et n'appelle jamais ***"INTERACTION"***.

```
ACTION INTERACTIVE

  VL : variables locales ;

  BOUCLER
   Calcul(VL) ;
   SI  Cond1(VL)  ALORS
     INTERACTION ;
   FIN SI
   SI  Continuer  ALORS
     CHOISIR Cond2(VL,INTERACTION.résultats)  PARMI

                R1 : A1(VL,INTERACTION.résultats) ;
                .
                Rn : An(VL,INTERACTION.résultats) ;
     FIN CHOIX
   FIN SI
  JUSQU' A  Abandon ou Cond3(VL,INTERACTION.résultats)
FIN AI
```

(figure III.5.)

Les Ai sont des actions (AT ou AI) qui ont accès aux Variables Locales et aux résultats de l'*INTERACTION* (coord., objets, scalaire, ...).

Les Ri sont les différents déroulements du traitement en fonction de conditions qui dépendent d'éventuels calculs sur les Variables Locales et/ou sur les résultats de l'*INTERACTION*.

Les Condi sont des conditions qui dépendent d'éventuels calculs sur les Variables Locales et/ou sur les résultats de l'*INTERACTION*.

Abandon est un booléen global initialisé par *INTERACTION*, dans tous les cas qui spécifient une volonté d'interrompre brutalement l'action en cours (Différés ou abandon explicite). Si *Abandon* est vrai, *Continuer* est faux.

Si Cond3 est toujours vérifiée, l'action est un traitement purement séquentiel, sinon c'est une itération.

Les Variables Locales peuvent prendre des valeurs qui dépendent des résultats de l'*INTERACTION*, ce qui implique que Cond1 peut dépendre de ces mêmes résultats.

Si Cond1 est fausse, Cond2 en tient compte pour ne pas accéder à des résultats d'une *INTERACTION* qui n'a pas eu lieu. De plus, dans ce cas, *Continuer* est forcément vrai.

La forme AI fait clairement apparaître la dualité qui existe entre le dialogue et le code. *INTERACTION* se sert du code par l'intermédiaire d'*APPEL*, tout en gérant de façon transparente les mécanismes de l'interface. Le code se sert d'*INTERACTION*, ce qui lui permet de contrôler en partie le flot d'entrées en orientant le déroulement de ses traitements en fonction du comportement de l'utilisateur. C'est en ce sens que le contrôle est à la fois externe et interne.

Bien que ce schéma d'ACTION ne soit pas complètement formel, il est à considérer comme un élément à part entière de notre modèle. Celui-ci est axé sur une Base de Données (structure de menus) que le code peut consulter explicitement au moyen de primitives d'accès (figurées par *INTERACTION.résultats*), et modifier implicitement grâce à la primtive unique *INTERACTION* (activation, désactivation, ...).

Donnons un exemple pratique de l'application C.A.O. afin de bien comprendre le principe. Nous utilisons un pseudo-code de l'option permettant de créer un cercle en fixant son centre et son rayon.

```
CERCLE

  X, Y, R : réel ;
       I : entier ;

  DEBUT

    INTERACTION(1) ;

    SI Continuer ALORS

      (X,Y)<-- COORD(1) ;
      INTERACTION(1) ;

      SI Continuer ALORS
        R<-- SCALAIRE(1) ;
        Créercercle(I,X,Y,R) ;
        Afficher(I) ;
      FIN SI

    FIN SI

  FIN CERCLE
```

(figure III.6.)

Créercercle est une primitive prédéfinie qui ajoute au modèle géométrique un cercle de centre (X,Y) et de rayon R en lui attribuant un numéro restitué dans I (globalisation).

Afficher est une primitive prédéfinie qui affiche à l'écran un objet de numéro I (localisation).

L'action ne manipule de façon directe que ses Variables Locales (X, Y, R, I), ce qui garantit l'intégrité et la transparence des différentes Bases de Données (Menus, Géométrie) du système qui fournit des Bases de Primitives d'accès.

Le **paramètre d'INTERACTION** permet aux primitives d'accéder aux résultats de la bonne interaction. Ceci introduit une notion de **numéro d'INTERACTION**. Une interaction est ainsi vue comme un objet de l'interface, d'une façon analogue à un objet du modèle géométrique. C'est ce qui explique que dans notre exemple on s'autorise le même numéro pour deux interactions dans la mesure où les résultats de la première sont mémorisés ((X,Y)<--COORD(1)) avant de faire allusion à la seconde. Nous aurions pu avoir :

```
CERCLE
  X, Y, R : réel ;
       I : entier ;
  DEBUT
    INTERACTION(2) ;
    SI Continuer ALORS
      INTERACTION(1) ;
      SI Continuer ALORS
        (X,Y)<-- COORD(2) ;
         R<-- SCALAIRE(1) ;
         Créercercle(I,X,Y,R) ;
         Afficher(I) ;
      FIN SI
    FIN SI
  FIN CERCLE
```

(figure III.7.)

qui montre que la numérotation n'est pas nécessairement liée à l'ordre dans la séquence des interactions.

Avant d'en voir les implications sur la modélisation, profitons de cet exemple simple pour le comparer à une approche plus classique. Supposons l'existence d'autres options :

- ZOOM pour agrandir un détail,
- INTER2OB pour créer un point intersection de deux objets.

Supposons que l'on désire autoriser un ZOOM durant la création d'un cercle, d'une part, et que le centre de ce cercle puisse être spécifié par un couple de coordonnées quelconques ou comme un point déjà existant ou comme intersection de deux objets, d'autre part. L'approche classique donnerait (à peu près) :

```
CERCLE

  X, Y, R : réels ; I : entier ; Arret : booléen ;
  DEBUT
    BOUCLER
      LOCALISER(X,Y) ;
      Arret<--vrai ;
      SI SELECTION(X,Y)=ZOOM
      ALORS
            ZOOMER;
            Arret<--faux;
      SINON
        SI SELECTION(X,Y)=INTER2OB
        ALORS
              CREERINTER2OB(X,Y) ;
        SINON
          SI DESIGNATION(X,Y)=point
          ALORS
                (X,Y)<--coord(point);
          FIN SI
        FIN SI
      FIN SI
    JUSQU'A Arret
     VALUATION(R);
     Créercercle(I,X,Y,R);
     Afficher(I) ;
  FIN CERCLE
```

(figure III.8.)

Alors que dans notre cas, le code reste celui que nous avons donné dans la mesure où :

- la primitive COORD restitue automatiquement :

  + les coordonnées du pointé si aucun objet n'est récupéré,
  + les coordonnées d'un point désigné directement ou implicitement créé,

- le concepteur précise ZOOM comme Immédiat et INTER2OB comme Local de CERCLE.

Cet exemple comparatif inspire quelques réflexions. On pourrait penser qu'il suffirait d'écrire des primitives de plus haut niveau à l'aide des primitives de base pour obtenir un résultat comparable avec l'approche classique. En fait, la similitude ne résiderait que dans certains aspects du code, en effet :

- la spécification de primitives de plus haut niveau serait à la charge du programmeur car celles-ci seraient particulières et différentes pour chaque traitement,

- la généralité de notre principe donne du dynamisme au contrôle du flot d'entrées. Par exemple, si l'on ne désire plus autoriser le ZOOM, il suffit de supprimer le lien de compatibilité qui existe entre ZOOM et CERCLE, et ne rien recompiler, alors qu'avec l'autre solution, il faut réécrire une primitive qui exclut le ZOOM, et recompiler. De même, si au lieu de ZOOM, on désire autoriser une demande d'AIDE, ou un REAFFICHAGE de la scène, ou des créations d'objets (SEGMENT, ARC, CERCLE ....) dont on pourrait désigner les intersections par la suite, il suffit de rendre toutes ces options compatibles (Immédiats) avec CERCLE. Si en plus d'un centre construit comme intersection, on veut un centre comme centre d'un autre cercle, il suffit d'ajouter un Local qui crée un tel point.

Les associations ne sont pas dépendantes de la sémantique même des Immédiats et des Locaux éventuels.

De plus, la compatibilité en soi n'apporte rien à la sémantique proprement dite d'une action. Elle ne doit donc pas s'exprimer dans le code. Par exemple, la façon dont on crée deux objets, pour désigner par la suite leur intersection comme centre, n'est pas significative dans la construction du cercle lui-même.

Un Local apporte un complément aux résultats d'une INTERACTION. Dans cette mesure, on peut dire qu'il permet de modifier le sens d'une action. Mais cela ne signifie pas pour autant qu'il doive apparaître dans la sémantique de base d'une action. Ce qui importe, c'est que l'on puisse récupérer un point, et non pas comment s'exprime le point en question.

Donnons un autre exemple pour fixer les idées à ce propos. Supposons un traitement qui récupère une liste d'objets, présents ou non sur la scène.

```
LISTEOBJ  { AI }
  DEBUT
    Arret<--faux ;
    BOUCLER
      INTERACTION(1) ;
      SI  Continuer
      ALORS
            SI  NBOBJ(1)<>0
          ALORS liste<--liste + OBJET(1) ;
          FIN SI
      FIN SI
    JUSQU'A  Abandon ou Arret
  FIN

STOP  { AT }
  DEBUT
    Arret<--vrai ;
  FIN
```

(figure III.9.)

Arret est un booléen global accessible par **LISTOBJ** et **STOP**.

STOP, **CERCLE**, **SEGMENT** sont spécifiés comme Locaux de **LISTOBJ** .

POINT, **ZOOM** sont spécifiés comme Immédiats de **LISTEOBJ**.

Pour **LISTEOBJ**, ce qui importe (sémantique), c'est de récupérer des objets et de les insérer dans sa liste jusqu'à décider de s'arrêter. Que les objets soient désignés directement, ou construits directement (**CERCLE**, **SEGMENT**), ou construits puis désignés (**POINT**), ne présente pas d'intérêt pour le sens de l'action. De même, l'action **STOP** précise le contexte en initialisant la condition d'arrêt. Que le menu Local associé s'appelle STOP, FIN, ARRET ou TRUCMUCHE n'a aucune importance.

Le fait de spécifier des menus Immédiats ou Locaux d'un autre exprime une volonté de modifier le contexte d'une action sans toucher à sa sémantique profonde, afin de procurer du dynamisme au code. Où se trouve la frontière entre syntaxe (contrôle du flot) et sémantique (code)? Cette question essentielle nous paraît plutôt subtile. C'est là un sujet de controverse qui est au centre de nombreuses approches. Notre solution, par son dynamisme, renforce le caractère dual du concept.

III.2. DEUXIEME VERSION

Au regard du paragraphe précédent, il est indispensable de compléter la structure de menus. De plus, nous devons préciser les choix logiciels concernant une application. Ceci conduira à une vue d'ensemble plus explicite encore du modèle.

III.2.1. PLACE DE L'INTERACTION DANS LE MODELE

Comme nous avons déjà pu l'entrevoir, l'INTERACTION n'est pas seulement une primitive de dialogue au sens procédural du terme. C'est également un concept, unique et transparent, caractéristique d'une action, et ce à plusieurs titres :

- elle gère tout ce qui est inhérent aux Immédiats et aux Locaux (comportements, aspects visuels, conséquences, ...),

- elle n'a pas de paramètres d'entrées-sorties. Elle interprète une intention et le code consulte ses résultats.

En fait, si l'on approfondit un peu, on constate que Locaux et Immédiats sont plus spécifiquement caractéristiques d'une INTERACTION, plutôt que de toute une action. Par exemple, pour l'action CERCLE, nous avions défini POINT comme Local de toute l'action. Ceci nous autorisait à le sélectionner au moment d'entrer le rayon. Ce qui peut paraître, non seulement inutile (voir dénué de sens), mais qui pose un problème lié au principe même d'un Local. En effet, si au lieu d'introduire la valeur du rayon, on sélectionne POINT, on sortira d'INTERACTION sans pouvoir initialiser la variable R.

Ces remarques ont plusieurs implications :

- les Immédiats et Locaux sont spécifiés pour une INTERACTION donnée qui les lie à l'action dans laquelle elle intervient,

- une action peut utiliser plusieurs INTERACTIONs dont les Immédiats et Locaux ne sont pas nécessairement les mêmes,

- il est impératif de savoir si un résultat possible a été ou non initialisé par INTERACTION au moment où on le consulte,

- une consultation n'intervenant pas forcément immédiatement après une INTERACTION, il faut conserver ses résultats,

- si lors de sa consultation, un résultat d'INTERACTION n'a pas été initialisé, la primitive qui y accède doit automatiquement permettre une INTERACTION en conséquence, et ce de façon transparente. Par exemple :

```
SCALAIRE(i)
 DEBUT
  TANT QUE  INTERACTION(i).scal non initialisé
       INTERACTION(i)
  FIN TQ
  SCALAIRE(i)<--INTERACTION(i).scal
 FIN
```

(figure III.10.)

Des traitements complexes peuvent utiliser des informations autres que les seuls résultats d'une INTERACTION (liées à la compatiblité ou autre).

De plus, le principe d'interprétation par défaut (cf. paragraphe I.3.5.) a montré l'existence d'autres renseignements relatifs à une INTERACTION (message).

Dans un souci de clarté, nous avons volontairement omis certains éléments concernant le comportement d'une INTERACTION. Jusqu'ici, ils n'étaient pas utiles à la compréhension des principes. Il n'en reste pas moins que nous les considérons également comme fondamentaux dans notre modèle.

Le premier de ces points se rapporte à la notion de touche de fonction au sens général. Une action peut donner une signification particulière à certains caractères tapés au clavier, et rediriger son traitement en conséquence. Par exemple :

```
  INTERACTION(1) ;
SI CARACTERE(1)='A' ALORS A1 ;
SI CARACTERE(1)='B' ALORS A2 ;
```

(figure III.11.)

Ce qui implique que l'INTERACTION l'interprète comme une intention atomique. Or, jusqu'à présent, notre principe considérait un caractère comme un élément d'expression. Il est donc nécessaire qu'une INTERACTION ait connaissance de ce renseignement pour passer le contrôle au code dès qu'un tel caractère intervient.

On peut noter la similitude fonctionnelle qui existe avec la notion de Local qui provoque également une sortie de l'INTERACTION, et qui peut avoir une incidence sur le déroulement du code. La différence étant qu'avant que l'INTERACTION ne se termine, un Local exécute un traitement. Du fait que l'on peut toujours spécifier explicitement un menu au clavier, il suffit de nommer un Local par un caractère spécial pour lancer automatiquement une action. La notion de touche de fonction est ainsi généralisée, devenant quasi équivalente à celle de Local, et profitant ainsi de toute la puissance du principe.

Le second point a trait au principe général de filtre. A savoir : préciser le rayon d'action d'une INTERACTION lorsqu'elle est amenée à retrouver des objets de la scène. Les objectifs étant :

- diminuer les temps de réponse, puisque la recherche se fait sur un ensemble restreint,

- éviter certaines ambiguïtés. Par exemple, si un pointé a lieu à proximité de deux objets O1 et O2 de types T1 et T2, et que l'on spécifie qu'on ne s'intéresse qu'à ceux de type T1, l'INTERACTION ne restituera que O1. L'utilisateur n'aura pas à lever l'ambiguïté apparente,

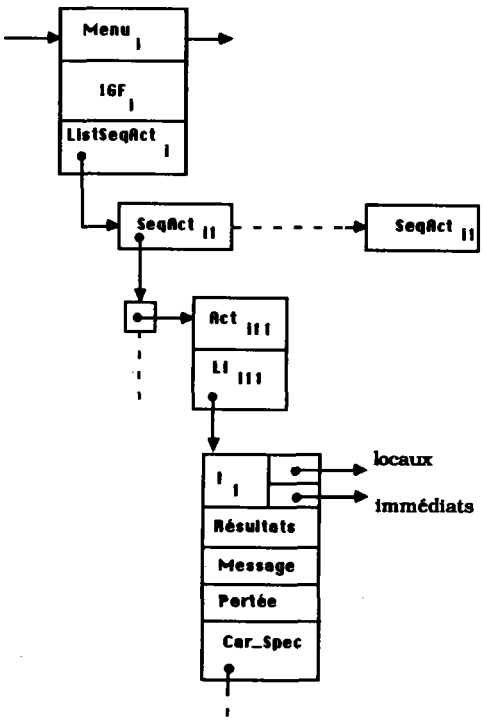
- contrôler l'exécution du code et garantir, dans certains cas, la consistance du traitement. Par exemple, soit :

```
BOUCLER
   INTERACTION(1) ;
JUSQU'A NBOBJ(1)>0
C1<--OBJET(1) ;
BOUCLER
   INTERACTION(1) ;
JUSQU'A NBOBJ(1)>0
C2<--OBJET(1) ;
S1<--SEGTG2CER(C1,C2) ;
```

(figure III.12.)

un traitement qui crée un segment tangent à deux cercles désignés interactivement. Si l'on limite l'INTERACTION aux seuls objets de type cercle, on reste en situation d'INTERACTION (contrôle) tant que l'on n'a pas récupéré un cercle et le code n'a pas à vérifier le type des objets récupérés puisque le filtre garantit qu'il s'agit forcément de cercles; ainsi la construction du segment est réalisable (consistante).

Tout cela nous a conduit à stocker les informations relatives à une INTERACTION. Il était naturel d'intégrer l'ensemble à la structure d'une action (cf. figure III.13.).



(figure III.13.)

RESULTAT est l'ensemble des résultats possibles pour une INTERACTION (X, Y, Z, NBOBJ, OBJET, SCALAIRE, ...). PORTEE exprime le filtre sur la recherche d'objets. CAR_SPEC sont les différentes touches de fonctions qui ne sont pas des Locaux. MESSAGE est un message associé à une INTERACTION. Il rend la récupération de coordonnées moins prioritaire qu'un scalaire (ou une chaîne). LISTSEQACT est une liste de séquences d'actions qui permet de définir des macro-menus dont les composants possèdent eux-mêmes une séquence d'actions (extension de la première version).

De façon analogue aux primitives de consultation, il existe des primitives d'initialisation (transparentes) qui permettent au code de contrôler la PORTEE, les CAR_SPEC, le MESSAGE, ... Par exemple :

```
T1<--cercle; T2<--segment;
C1<--'A'; C2<--'B';
BOUCLER
  PORTEE(1,T1,T2);
  CAR_SPEC(1,C1,C2,'D');
  INTERACTION(1);
  SI CARACTERE(1)='A' ALORS A1; FIN SI
  SI CARACTERE(1)='B' ALORS A2; FIN SI
  SI CARACTERE(1)='D' ALORS
    C1<--'C';
    C2<--'E';
  FIN SI
  SI CARACTERE(1)='C' ALORS T1<--point; FIN SI
  SI CARACTERE(1)='E' ALORS
    C1<--'A';
    C2<--'B';
    T2<--arc;
  FIN SI
JUSQU'A  Abandon
```

(figure III.14.)

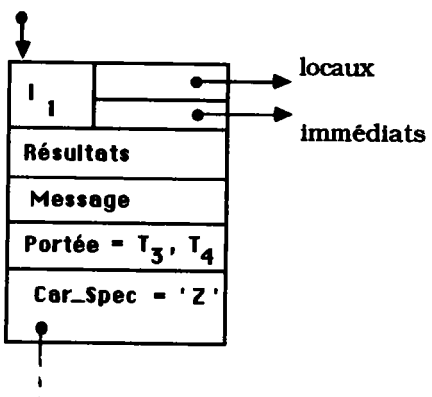
Ce qui n'empêche pas de les initialiser à la conception de l'interface (en accord avec le programmeur) dans le cas où il n'est pas utile que le code les modifie. Par exemple, au lieu de :

```
PORTEE(1,T3,T4);
CAR_SPEC(1,'Z');
INTERACTION(1);
SI CARACTERE(1)='Z' ALORS A1;
```

on aurait, le code :

```
INTERACTION(1);
SI CLAVIER(1) ALORS A1;
```

et la structure :



(figure III.15.)

qui permettrait de modifier la valeur de la touche de fonction sans modifier son rôle, et sans recompiler.

De même, le code pourrait contrôler les Immédiats et Locaux grâce à des primitives analogues. Cette approche semble offrir de nombreuses possibilités. C'est pourquoi, les exemples ci-dessus sont à considérer comme des illustrations des principes.

Nous avons introduit une certaine rigueur dans la façon de concevoir une application :

- le code ne doit exprimer que la sémantique profonde d'une action,

- l'interface gère les dialogues possibles en exprimant les relations éventuelles entre options.

Nous pensons que la dualité de notre approche est un atout dans la mesure où elle permet de séparer les rôles de chacun pour une mise en oeuvre dynamique, tout en autorisant des dialogues complexes grâce à une utilisation efficace des concepts. Mais qui peut le plus peut le moins; rien n'empêche le programmeur d'écrire du code qui tende à contrôler le dialogue, et ainsi, perdre une grande partie des avantages qu'apportent l'uniformité, le dynamisme et la transparence du modèle.

III.2.2. CHOIX LOGICIELS

L'organisation du code d'une application doit permettre:

- la modularité,
- une exécution rapide,
- des traitements récursifs faciles à spécifier et à gérer (par exemple, il peut être intéressant de pouvoir créer un segment pendant la construction d'un autre segment [le code correspondant n'est pas toujours évident à mettre en oeuvre]),
- la définition de structures de données et de contrôle puissantes et fiables.

C'est pourquoi nous nous sommes dirigés vers l'utilisation de modules de fonctions. Quelle que soit la façon d'obtenir le code effectif, celui-ci finit par s'exprimer dans un langage évolué compilé (Pascal, C, ...).

Ce choix présente des avantages certains qui sont liés à la mise à profit des propriétés des langages :

- puissance de description algorithmique,
- capacité à définir et à manipuler des structures de données complexes,
- cela ne nécessite pas l'écriture d'un interpréteur et, surtout, évite son utilisation à l'exécution, d'où gain de temps et de place,
- récursion gérée par la pile d'exécution, d'où sauvegarde automatique et consistante de l'environnement,
- les notions de variables locales et globales sont intégrées,
- la modularité est au centre de leur philosophie.

Il est clair que la limite majeure des langages compilés est leur manque de dynamisme comparés à des langages entièrement interprétés. Néanmoins, nous avons la conviction que notre modèle y remédie dans une large mesure. En effet, la structure de menus, la primitive INTERACTION, la primitive APPEL et le NOYAU sont des éléments compilés qui constituent ce que l'on peut considérer comme un interpréteur "standard" du dialogue. Il s'agit d'une solution intermédiaire qui nous semble un compromis efficace.

Un module est donc un ensemble de fonctions (ou procédures) internes susceptibles d'être associées à un nom d'action externe, et donc d'être lancées par APPEL. Sa forme générale est :

```
MODULE M1

  VG = variables globales du module;
  FCT1
    VL1 = variables locales de la fonction 1;
    CORPS1;
   .
   .
   .
  FCTn
    VLn = variables locales de la fonction n;
    CORPSn;

  FCT_LOC1
    VL_LOC1 = variables locales de la fonction loc1;
    CORPS_LOC1;
   .
   .
  FCT_LOCp
    VL_LOCp = variables locales de la fonction locp;
    CORPS_LOCp;

FIN M1
```

(figure III.16.)

Chacun des modules est lié à un concept différent des autres. Il regroupe les actions associées à un menu et à ses Locaux utilisant les mêmes Variables Globales. Cela signifie que le code de l'action d'un Local peut également se trouver dans un autre module que celui du concept principal, dans la mesure où il n'utilise pas les Variables Globales de celui-ci.

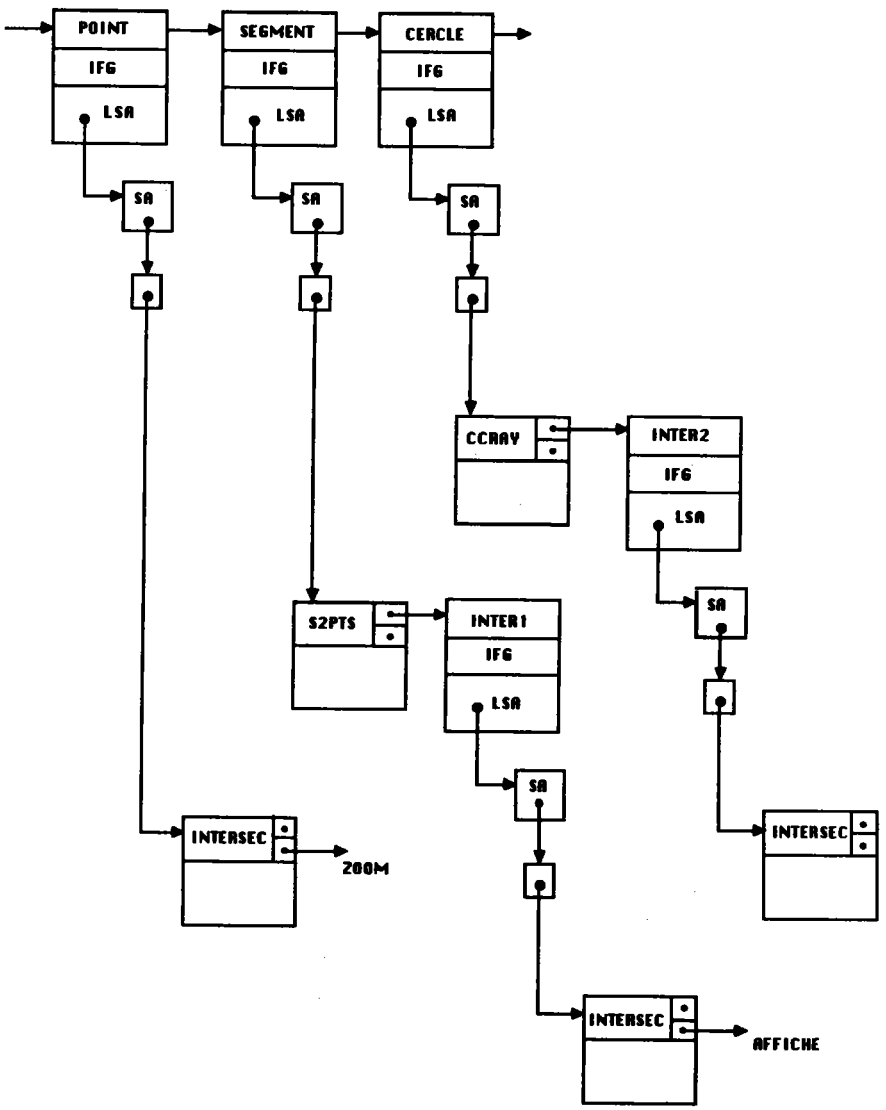
Ce qui permet de partager le code de Locaux fonctionnellement différents, dans la mesure où ils correspondent à des éléments distincts de la structure de menus, mais dont l'action associée est la même. Autrement dit, la référence au code peut être plusieurs fois dupliquée, alors que le code effectif n'est défini qu'une seule fois.

Prenons un exemple :

```
MODULE  POINT     MODULE  SEGMENT     MODULE CERCLE

  VGp;              VGs;                VGc;
   .                 .                   .
   .                 .                   .
  POINTER           .                   .
  INTER2OB          SEG2PTS             CERCR
   .                 .                   .
   .                 .                   .
FIN POINT         FIN SEGMENT         FIN CERCLE


MODULE  ZOOM      MODULE  AFFICHAGE

  VGz;              VGa;
   .                 .
   .                 .                   .
  ZOOMER            AFFICHER
   .                 .                   .
   .                 .                   .
FIN ZOOM          FIN AFFICHAGE
```

(figure III.17.)

avec la structure de menus :



(figure III.18.)

qui montre que le code n'est pas dupliqué, mais également que ce même code peut localement se comporter différemment. Entre autres, la construction d'une intersection peut autoriser un ZOOM dans le contexte d'un POINT, un AFFICHAGE dans le contexte d'un SEGMENT, et rien du tout dans celui d'un CERCLE.

Il faut remarquer (entre autres) que le principe permet de réaliser une même fonction tout en plaçant le menu associé à des endroits différents (à l'écran) suivant le contexte. Par exemple INTER2OB peut être situé à proximité de POINTER pour un point, de SEG2PTS pour un segment, ..., et ainsi renforcer la continuité visuelle.

Donnons un exemple d'utilisation des Variables Globales.

```
MODULE  LISTOBJ

 VG = Arret ;

 LISTSEG

   Arret<--faux
   BOUCLER
     .
     .
     .
   JUSQU'A  Abandon ou Arret

 FIN LISTSEG


 LISTPOINT

   Arret<--faux
   BOUCLER
     .
     .
     .
   JUSQU'A  Abandon ou Arret

 FIN LISTPOINT
   .
   .
   .
   .
   .
   .
 STOP
   Arret<--vrai;
 FIN STOP

FIN LISTOBJ
```

(figure III.19.)

Chaque fonction correspond à un menu différent. Le menu associé à STOP est spécifié comme Local des menus associés aux fonctions LISTSEG, LISTPOINT, ... Ainsi, sa sélection arrêtera la saisie d'objets dans tous les cas.

Le code d'une fonction définie dans un module peut faire appel à une fonction définie dans un autre module.

Les modules peuvent être compilés séparément.

L'ensemble des primitives d'accès aux différents modèles (INTERACTION, géométrie, ...) constitue des modules pré-définis compilés une fois pour toute (bases de primitives).

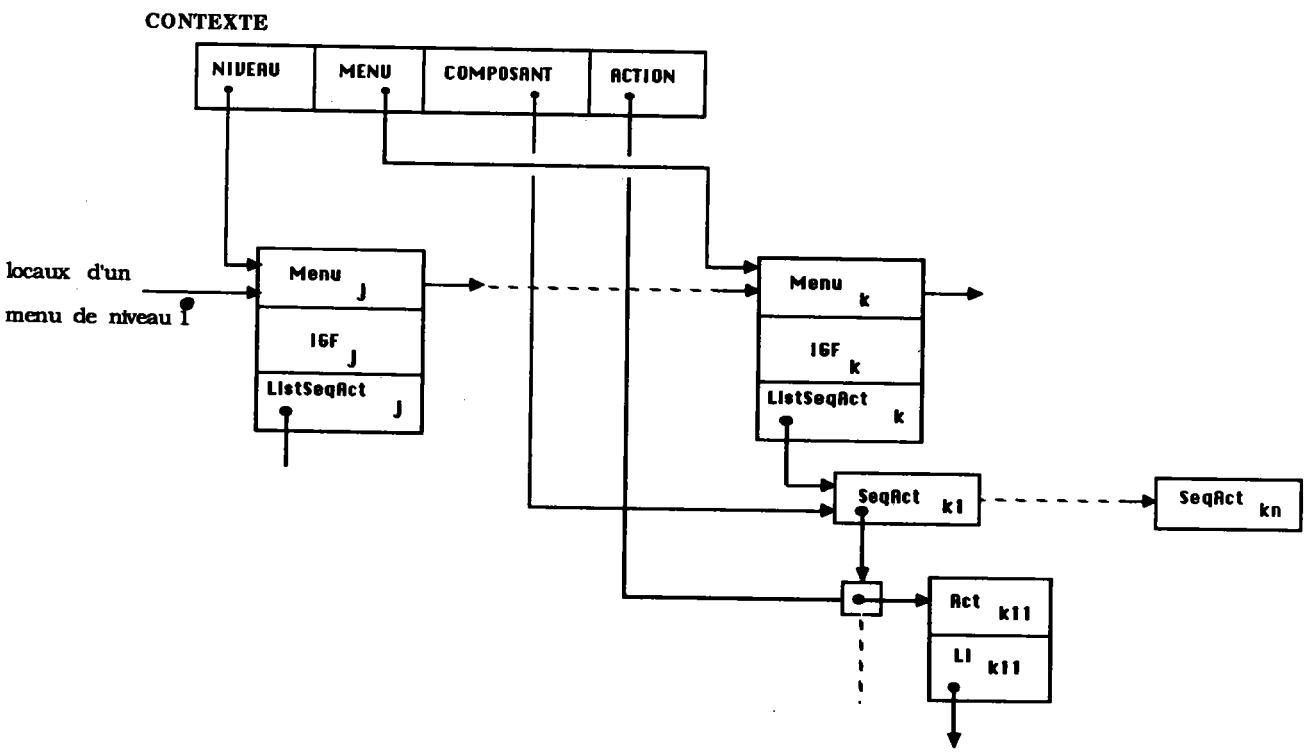
Le paragraphe suivant introduit de nouveaux éléments du modèle permettant d'en donner une version plus complète.

III.3. "DERNIERE" VERSION

III.3.1 STRUCTURE COMPLEMENTAIRE

Il est un point que nous n'avons pas encore abordé. Il s'agit du lien qui existe entre une action effectivement en cours et l'ensemble des informations qui lui sont rattachées. Entre autres, pour pouvoir retrouver les Immédiats et les Locaux à l'exécution, la primitive INTERACTION doit connaître l'action qui l'appelle, ainsi que le niveau courant. Puisque son seul paramètre est un numéro, il n'existe pas de référence explicite à l'action. Il faut donc conserver et gérer ce type d'informations de façon transparente.

Ce qui conduit à utiliser une structure globale dont seule l'interface connaît l'existence. Appelons-la CONTEXTE en cours, c'est-à-dire l'action en cours, donc également le composant où elle se trouve, et par conséquent le menu dont elle est issue, ainsi que le niveau où se situe le menu. Soit :

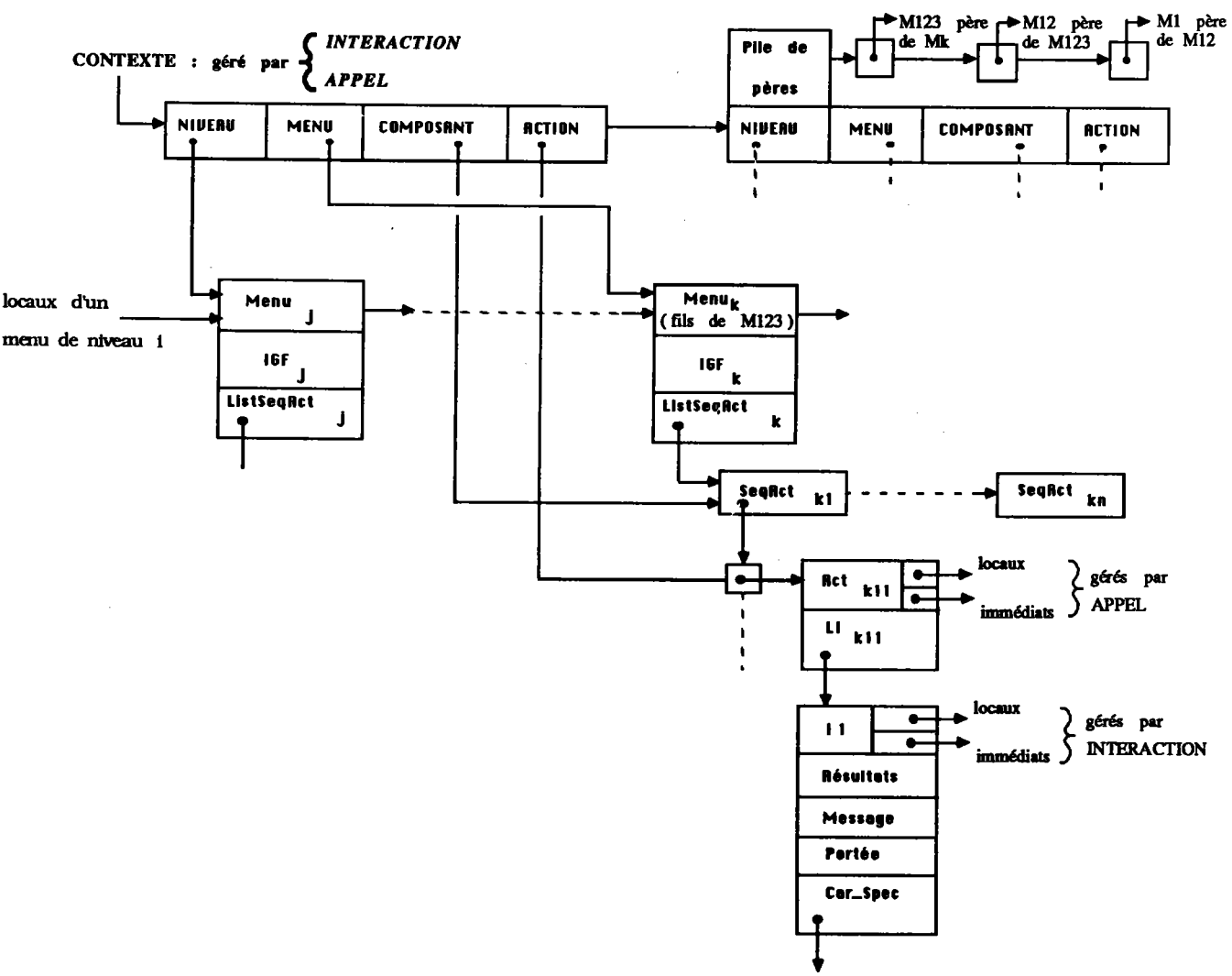


(figure III.20.)

Le menu courant peut se trouver à l'un quelconque des niveaux de la structure de menus (ici, il est au niveau i+1).

De plus, on peut avoir une imbrication d'appels à INTERACTION en sélectionnant une cascade d'Immédiats et/ou de Locaux. Si le CONTEXTE courant est relatif à un menu i et que l'on sélectionne menu j, un de ses Immédiats, le CONTEXTE courant deviendra relatif à

ce dernier. Au retour de son exécution, l'INTERACTION qui l'aura appelé aura de nouveau besoin des informations relatives à menu i. Par conséquent, il est indispensable de conserver un CONTEXTE qui en active un autre. La structure est donc une pile de contextes imbriqués. Le sommet de pile figure le CONTEXTE courant. D'où les extensions suivantes :



(figure III.21.)

Il est important de souligner que la pile est un élément à part entière du modèle. Son existence est totalement ignorée de l'application qui n'y a pas accès. Il s'agit d'un empilement fonctionnel. En tant que tel :

- il n'est pas lié à la pile d'exécution, donc pas lié au code (langage utilisé),

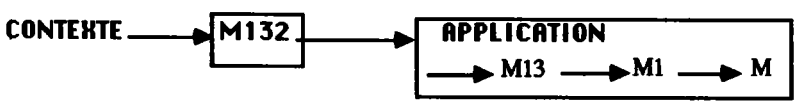
- il n'intervient pas dans les récursions proprement algorithmiques, ne sauvegarde pas l'environnement (VL et VG) du code.

La pile est gérée tantôt par APPEL, tantôt par INTERACTION, suivant le CONTEXTE.

Une descente dans une arborescence ne provoque pas d'empilements de contextes. C'est un contexte unique, alors que des appels successifs d'Immédiats et/ou de Locaux constituent des imbrications de contextes. Ainsi, si l'on a l'arborescence suivante :

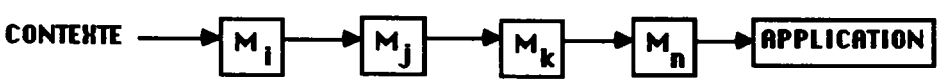
M
  M1
    M11
    M12
    M13
      M131
      M132
  M2
  M3
  M4

et que l'on descende jusqu'à M132, le CONTEXTE sera :



(figure III.22.)

Alors que si l'on a Mi Immédiat de Mj, Immédiat de Mk, Local de Mn, et que l'on descende jusqu'à Mi, le CONTEXTE est :



(figure III.23.)

Le niveau courant est référencé par le premier élément d'une liste de menus Locaux d'un autre (de l'action ou d'une de ses INTERACTIONs).

Il arrive souvent que les différentes INTERACTIONs d'une action possèdent des Immédiats et des Locaux en commun. Dans ce cas, ceux-ci peuvent être spécifiés pour l'action, plutôt que pour chaque INTERACTION.

Les figures précédentes montrent que la pile n'est jamais vide. En effet, il est un CONTEXTE particulier toujours actif. Ce dernier n'est autre que l'application elle-même. Ce CONTEXTE fait référence à un niveau 0 abstrait limité à un seul menu. Son action est le NOYAU lui-même. Ses Locaux sont les menus de niveau 1 qui apparaissent quand on lance l'application. Ils sont définis pour l'action puisqu'elle ne possède qu'une INTERACTION. De plus, ils sont implicitement Différés les uns des autres. L'action NOYAU n'a pas d'Immédiats car elle est seule à son niveau et qu'il n'y a pas de niveaux au dessus (pour l'instant....). Le menu associé au NOYAU n'est pas visualisé à l'écran (pas d'IGF).

Il semble clair que notre modèle intègre complètement les principes d'un système informatique en général. En effet, lancer une application n'est autre que sélectionner une option du système d'exploitation dont l'ensemble des commandes constitue le niveau 0.

Cette remarque est essentielle car elle signifie que le fait d'accéder à une fonction système à partir d'une application rentre dans le schéma de gestion des Immédiats (et Différés). Par exemple, appeler un compilateur (ou un éditeur, ou ...) peut à volonté être considéré comme un désir de quitter l'application et d'entrer directement en compilation (ou en édition, ou ...) Différée, ou de suspendre l'application et d'y revenir automatiquement après compilation (ou édition, ou ...) Immédiate. Cela accentue le caractère général de notre modèle.

III.3.2. OPTIONS PREDEFINIES

Nous avons considéré que le fait de vouloir abandonner une action, sans pour autant en lancer une autre, était une fonctionnalité générale indépendante de l'action en cours, donc de l'application. Il en va de même en ce qui concerne la sélection d'une option dont l'objet est de fournir des explications quant à l'utilisation de l'action (ou l'INTERACTION) en cours.

C'est pourquoi nous proposons d'enrichir le modèle par la présence de deux options que le programmeur n'a pas à décrire et qui sont automatiquement prises en compte par l'interface. Appelons-les QUITTER et AIDER. Quelle que soit l'application, elles seront présentes à l'écran et accessibles à tout moment. La gestion des informations relatives à chacune d'elles trouve donc une place dans le modèle.

L'option "QUITTER :

- le menu correspondant figure toujours dans la liste des menus du niveau 1,
- le menu est automatiquement considéré comme Immédiat de tout autre menu de l'application,
- l'action associée est définie une fois pour toute. Elle est automatiquement prise en compte par APPEL,

- l'action consiste à demander une confirmation de la volonté d'abandon, et à initialiser ce qui est nécessaire à la bonne gestion de la décision prise (pile, abandon, ...),

- la demande de confirmation est une INTERACTION, avec toutes les propriétés que cela implique,

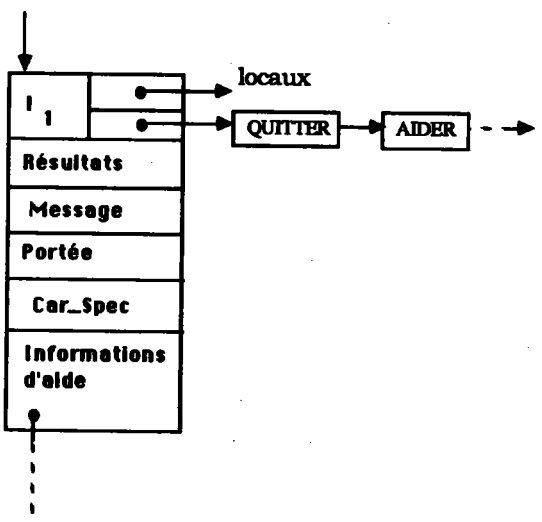
- son comportement est complètement transparent.

L'option "AIDER" :

- a les mêmes propriétés de définition et de gestion automatiques que "QUITTER" (au niveau 1, toujours Immédiat, connue d'APPEL, ...),

- son action consiste à retrouver des informations explicatives de l'INTERACTION en cours. Elle y accède par la connaissance du CONTEXTE (pile),

- les informations sont toujours présentées de la même façon à l'utilisateur. Elles sont spécifiées dans la structure générale, dans la définition d'une INTERACTION.



(figure III.24.)

- ce nouveau champ de la structure pourra être initialisé suivant les mêmes principes que les autres (concepteur ou programmeur). A l'heure actuelle, il n'est pas encore implanté. Il y a donc plusieurs possibilités :

  + un texte explicite,
  + une référence à un fichier,
  + une référence à une fonction complexe permettant de personnaliser l'aide,

  .....

Plus que leur implantation proprement dite, ce qui nous paraît important concernant ces deux options, est leurs places et rôles dans le modèle. Elles sont :

- indispensables (l'expérience le prouve),
- transparentes (déchargent le concepteur et le programmeur),
- uniques (accroît la consistance de leur utilisation),
- générales (valables pour toute application).

Nous les considérons donc comme caractéristiques de notre modèle.

Il faut remarquer que leur existence n'empêche pas une application de posséder des options qui joueraient des rôles analogues, plus ponctuels, voir plus complets. Il suffirait de définir des Locaux ou des Immédiats plus spécifiques.

Les conséquences de leur utilisation sont constantes et doivent être connues. En tant qu'Immédiat, l'option AIDER suspend l'interaction en cours, donne des renseignements, et rend le contrôle à l'INTERACTION. Son utilisation est donc sans danger. Par contre, l'option QUITTER suspend l'INTERACTION en cours, et rend le contrôle :

- à l'INTERACTION, si le désir d'abandonner n'est pas confirmé (comportement normal d'un Immédiat),
- au code, si l'abandon est confirmé. Le booléen global "Abandon" devient vrai et le code s'en sert suivant ses besoins. Les différentes possibilités pour le déroulement du code sont de la responsabilité du programmeur, ce qui peut introduire des éléments d'inconsistance dans l'application. Par exemple, c'est au programmeur de décider si dans ce cas il conserve des résultats intermédiaires calculés avant l'interruption, ou s'il les "défaits". Par contre, une certaine consistance demeure dans le dialogue :
  + le principe est toujours le même, que ce soit à cause d'un Différé (implicite) ou de QUITTER (explicite). C'est toujours et uniquement le booléen global "Abandon" qu'il faut considérer,
  + ce qui doit être "défait" concernant les CONTEXTEs (dépilement, retour au bon niveau de contrôle, état du dialogue, ...) est automatique.

III.3.3. MECANISMES PREDEFINIS

L'utilisation possible d'arborescences de menus mène à des activations successives de sous-menus, suivant un principe "standard". Quel qu'il soit, tout menu possédant des fils est associé à une action qui est toujours la même, indépendamment de sa nature et de celle de ses fils.

Ce mécanisme prend donc sa place dans le modèle :

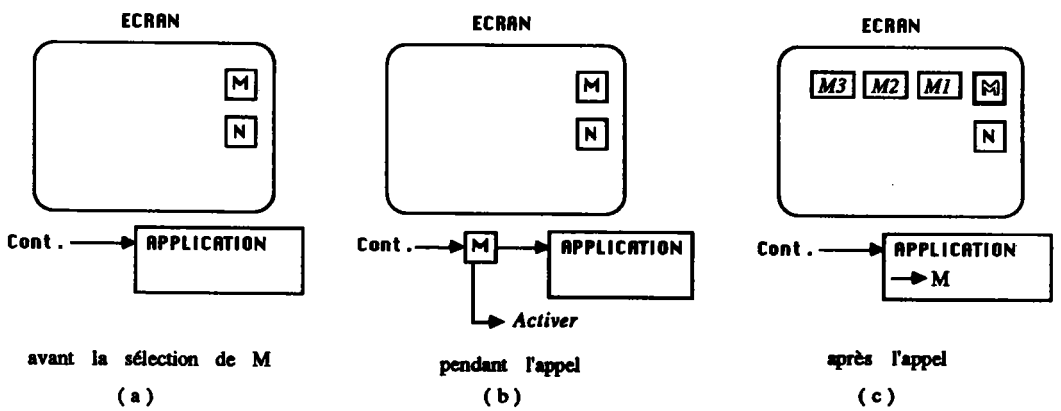
- une action pré-définie ACTIVER est automatiquement associée à un menu père,

- son nom suffit à savoir ce qu'il faut entreprendre si un père est sélectionné. L'activation de sous-menus est prise en charge par la primitive APPEL,

- son nom suffit à considérer les Locaux de l'action comme des sous-menus,

- l'action n'est liée à aucun code application effectif,

- un menu père n'apparait dans l'empilement des CONTEXTEs que le temps d'activer ses sous-menus :



(figure III.25.)

Les sous-menus sont stockés dans la liste des Locaux de l'action ACTIVER car ils répondent à la définition générale de menus, mais ils sont fonctionnellement différents car ils restent actifs après l'exécution de l'action ACTIVER qui est terminale.

La plupart des options d'une application sont utilisées de façon répétitive. Par exemple, le fait de sélectionner CERCLE signifie souvent que l'on désire construire plusieurs cercles. Or le but de l'action associée "Cercle" est de créer un seul cercle. La réitération du processus n'est pas caractéristique de la sémantique profonde de l'action et ne doit donc pas en être dépendante (ne doit pas apparaître dans le code).

En conséquence, nous avons considéré que toute action (excepté ACTIVER) associée à un menu boucle automatiquement. Autrement dit, on reste dans le concept choisi jusqu'à en choisir un autre.

Le fait de choisir un Immédiat conduit à une situation particulière. Intuitivement, l'importance du concept associé à un menu n'est pas la même suivant que le menu est sélectionné dans le contexte global ou comme Immédiat d'un autre. Par exemple, si l'on sélectionne CERCLE au niveau global, il est le centre d'intérêt (concept) principal, et en tant que tel, on reste dans le contexte (on réitère) jusqu'à choisir un autre centre d'intérêt (Différé). Par contre, si l'on sélectionne CERCLE comme Immédiat, c'est que le centre d'intérêt principal est le contexte dans lequel on a choisi CERCLE. L'importance du concept CERCLE est ponctuellement minimisée. Nous avons traduit cette notion par le fait que, dans ce cas, on ne réitère pas l'action la moins importante (un Immédiat ne boucle pas).

III.3.4. REVISION DES PRIMITIVES

Les primitives APPEL et INTERACTION doivent être modifiées. Compte tenu de tout ce qui a été dit jusqu'à présent, nous donnons les schémas de principe "quasi définitifs" des mécanismes mis en place.

```
APPEL (option)

 DEBUT
   Empiler le CONTEXTE ;
   BOUCLER
     BOUCLER sur CONTEXTE.ListSeqAct
       BOUCLER sur ListSeqAct.SeqAct
         Activer SeqAct.action.Locaux ;
         Visualiser SeqAct.action.Immédiats ;
         SI action=QUITTER
         ALORS
               Quitter ;
         SINON
           SI action=AIDER
           ALORS
                 Aider ;
           SINON
             SI action=ACTIVER
             ALORS
                   Activer ;
             SINON
                   SUIVANTCAS(action) ;
             FIN SI
           FIN SI
         FIN SI
         Revisualiser SeqAct.action.Immédiats ;
         SI action<>ACTIVER
         ALORS
               Activer SeqAct.action.Locaux ;
       JUSQU'A Fin SeqAct ou Abandon
     JUSQU'A Fin ListSeqAct ou Abandon
     SI Différé
     ALORS
           option<--menu sélectionné ;
   JUSQU'A Immédiat ou Activer
           ou Abandon ou AIDER
   Dépiler le CONTEXTE ;

FIN APPEL
```

Remarques :

- "Empiler" peut dépendre du CONTEXTE en cours (actuel sommet) avant empilement. Notamment, cela tient compte d'une éventuelle arborescence (désactivation des sous-menus) ou de la provenance d'un Différé (mise à jour de l'état)...

- l'activation des sous-menus est un cas particulier de l'activation des Locaux,

- "Dépiler" peut dépendre du CONTEXTE en cours (actuel sommet) avant dépilement. Notamment cela tient compte d'une arborescence dans le contexte précédent (nouveau sommet) ou des conséquences d'un abandon...

- le fait de sélectionner un des sous-menus d'un père désactive tous ses sous-menus (ils disparaissent pendant l'exécution du sélectionné). Au retour de celui-ci, les autres (du même père) sont réactivés (réapparaissent) automatiquement avec lui,

- "Activer" met à jour l'arborescence du CONTEXTE précédent,

- QUITTER force les conditions d'arrêt des différentes boucles,

- "SUIVANTCAS(action)" lance le code associé à l'action. Bien que "Quitter", "Aider" et "Activer" pourraient s'y trouver, nous les avons volontairement sortis pour marquer leur rôle dans le modèle. De plus, la fonction SUIVANTCAS peut être compilée séparemment d'APPEL. Ce qui signifie qu'une profonde modification de l'application ne nécessitera ni modification, ni recompilation d'APPEL. Elle se traduira par des retouches de SUIVANTCAS, c'est-à-dire uniquement de la correspondance nom externe d'action-nom interne du code.

```
INTERACTION (i)

do : entier; Différé1 : booléen ;

DEBUT

  Différé1<--faux ; Différé<--faux ;
  Activer Locaux de i ; {contexte}
  Visualiser Immédiats de i ; {contexte}

  BOUCLER

    SI Différé1=faux
    ALORS
          Interpréter l'intention ;
    SINON
          Différé<--vrai ;
    FIN SI
    SI un menu du CONTEXTE courant
    ALORS
        SI Local
        ALORS
              do<--dernierobjet ;
              APPEL(menu) ;
              Différé1<--Différé;
              Différé<--faux;
              SI do<>dernierobjet
              ALORS
                INTERACTION(i).obj<--dernierobjet;
              FIN SI


        SINON
              Immédiat<--vrai ;
              APPEL(menu) ;
        FIN SI
    FIN SI
  JUSQU'A Validation ou Abandon

  Revisualiser Immédiats de i ; {contexte}
  Désactiver Locaux de i ; {contexte}

FIN INTERACTION
```

Remarques :

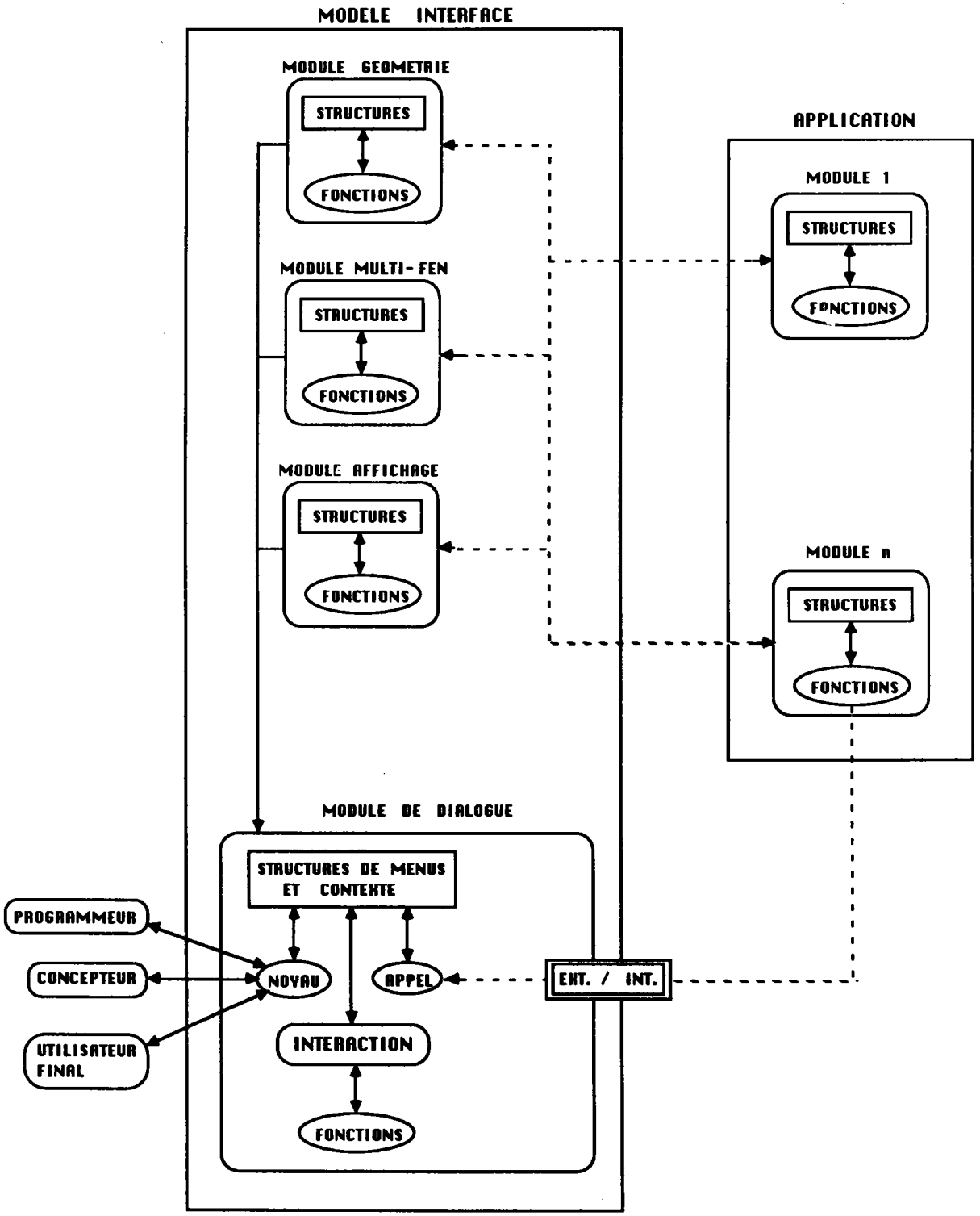
- comme APPEL, c'est par le biais de la pile des CONTEXTEs qu'INTERACTION accède aux informations qui lui sont nécessaires (Locaux, Immédiats, ...),

- "Interpréter" tient compte de l'environnement multi-fenêtré. Celui-ci est interactivement configurable à volonté et offre de nombreuses possibilités qui augmentent la convivialité du système (cf.Annexe 2). INTERACTION gère automatiquement la position courante de la souris et restitue toujours des coordonnées en accord avec ce que l'utilisateur perçoit de la scène.

Nous avons tenu à ne donner que des schémas de principe de ces primitives afin de ne pas compliquer les explications. Le but était de visualiser les mécanismes et les relations entre composants du modèle. Le paragraphe suivant donne une vue d'ensemble de l'architecture logicielle.

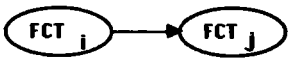
III.4. SCHEMA GENERAL

Nous proposons une figure de synthèse qui combine les différents modules du système, rappelant ainsi ses fondements.

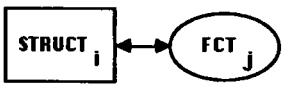


(figure III.26.)

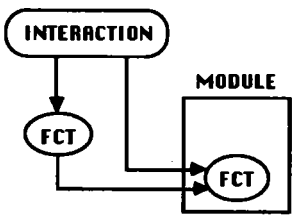
Légende



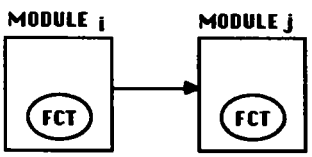
: les fonctions de FCT_j peuvent se servir de celles de FCT_i.



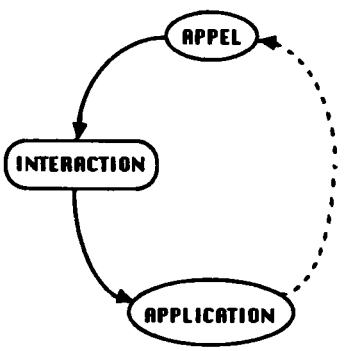
: les fonctions de FCT_j peuvent manipuler (consultation / modification) les structures de $STRUCT_i$.



: vue à la fois comme une
- primitive, car utilisable
- structure, car accessible.



: le $MODULE_j$ s'appuie sur le $MODULE_i$.



: montre la récursivité possible.

(figure III.27.)

Les paragraphes précédents ont fourni un aperçu de ce qu'était l'application C.A.O. développée dans un tel environnement. La quatrième partie présente deux applications destinées respectivement au programmeur d'application et au concepteur d'interface.

QUATRIEME PARTIE

DEUX UTILISATEURS PRIVILEGIES

Cette partie vise à montrer les avantages de notre approche concernant l'intégration du concepteur et du programmeur (prototypage, maintenance, ...).

Nous avons développé une maquette d'application destinée au concepteur d'interface. Cette application de SACADO fait l'objet du premier paragraphe. On pourra trouver des exemples d'utilisation en Annexe 3.

Par ailleurs, il semble important d'entrevoir des solutions destinées au programmeur, et qui s'appuient sur les principes énoncés jusqu'ici. Le deuxième paragraphe fait état d'un certain nombre de propositions qui vont dans ce sens, et qui montrent différentes ouvertures possibles.

IV.1. UNE REPONSE AU CONCEPTEUR

Ainsi que nous l'avons partiellement abordé dans la troisième partie, le rôle du concepteur d'interface est d'initialiser la structure de menus de notre modèle. Nous devons donc lui fournir un outil adapté à sa tâche. Fidèles à notre objectif de départ, nous avons considéré que cet outil n'est rien d'autre qu'une application dont l'utilisateur final est le concepteur lui-même. A ce titre, cette application se doit de répondre aux exigences générales que nous avons définies. Ce qui signifie que c'est une application interactive qui fonctionne dans le cadre de notre modèle. Elle se présente comme toute autre application et bénéficie de tous les concepts que nous avons introduits. Nous n'entrerons pas dans le détail de toutes les options qu'elle propose.

Le principe est le suivant. Au travers d'un dialogue que nous avons essayé de rendre le plus souple et le plus agréable possible, le concepteur manipule les objets de son monde : des vues externes des différents champs de la structure de menus. Il peut ainsi dynamiquement placer des options à l'écran, leur associer des actions (ou des séquences d'options), établir des liens de compatibilité, définir des hiérarchies, tester sa conception, prévoir tout ce qui est inhérent à un environnement multi-fenêtré, ...

Lorsqu'une session de travail se termine, les informations qui ont été ainsi spécifiées sont stockées dans un fichier dont le concepteur ignore la syntaxe. Le concepteur pourra reprendre son travail en commençant une session par le chargement du fichier (une des options). Une interface peut être de cette façon développée par touches successives. Ce qui encourage une conception descendante de prototypes utilisables rapidement et qui peuvent être affinés jusqu'à donner le résultat final.

Il est clair que le fichier qui définit le dialogue même de cette application a été écrit à la main. De plus, les actions de cette application accèdent aux différentes structures. Autrement dit, cette application est forcément dépendante de notre modèle. Ceci était inévitable. Mais il nous paraît important de souligner que ses actions utilisent la primitive INTERACTION et y accèdent suivant le schéma

donné dans la caractérisation d'une action. En fait, la seule chose qui la différencie d'une autre application, c'est son domaine proprement dit. Ceci est essentiel à plusieurs titres :

- c'est un exemple complexe non C.A.O. qui renforce la généralité de notre modèle. C'est un élément de validation de nos principes,

- tout ou partie de cette application peut être intégré à une autre dont le domaine est radicalement différent, ce qui accroît les possibilités de dynamisme à l'exécution,

- dès lors que ses fonctionnalités de base existent, elle peut s'utiliser pour s'auto-étendre, ce qui signifie que les parties écrites à la main se réduisent au minimum.

En dehors de l'outil puissant qu'elle représente, elle nous a permis d'améliorer et de valider notre approche. En effet, les dialogues complexes qui sont mis en oeuvre ont nécessité l'utilisation optimale des propriétés du modèle. Par exemple, des définitions récursives du type : le menu M1 a pour Local le menu M2 défini comme étant un macro-menu à un seul composant qui est M1 lui-même, auraient été difficiles à seulement imaginer dans le cadre d'une application plus simple. Or, l'exemple ci-dessus introduit des boucles dans la structure des menus, posant par là même le problème de la gestion correcte d'une telle situation. Ceci nous a permis de constater que notre principe de macro-menus fonctionnait dans sa généralité, mais qu'il fallait détecter les boucles de ce genre pour maintenir cohérent l'aspect externe et le fonctionnement des menus concernés. Nous avons ainsi pu combler certaines lacunes et corriger des erreurs d'implantation qu'il nous aurait été difficile de tester autrement.

Dans le même ordre d'idées, l'outil nous a permis d'écrire des dialogues fictifs mettant en jeu de nombreux niveaux de Locaux (ou des arborescences de plusieurs générations). Nous avons ainsi pu constater des limites matérielles :

- espace utilisable par la structure de menus,
- limite de la pile d'exécution dans l'utilisation d'options récursives.

Ces tests nous ont suggéré certains choix de structures et de vérification des contraintes d'intégrité qui s'y rattachent (nombre de menus définissables, ...).

Des fonctionnalités utiles à la spécification de dialogues complexes mettant à profit nos principes ont été implantées. La seule réelle expérience que nous ayons de son utilisation est la spécification du dialogue de l'application C.A.O. 2D qui nous a servi de support jusqu'ici et qui a été développée par ceux-là même qui ont défini les principes de base de notre modèle. Nous sommes assez optimistes dans la mesure où d'autres membres de l'équipe vont pouvoir s'en servir de façon objective pour ajouter des fonctionnalités (3D, surfaces, ...) qui sont actuellement testées indépendamment de notre architecture.

Par ailleurs, à notre connaissance, peu d'expériences similaires ont été abordées [ROA 82][KAM 83][BUX 83][COU 86]. De ce point de vue, il nous est difficile de nous situer par rapport à d'autres. Il semblerait que la plupart des auteurs proposent des outils qui nécessitent la connaissance de syntaxes précises. En tout cas, comme nous le verrons dans la cinquième partie, les principes mêmes de certaines approches compromettraient la qualité des outils destinés au concepteur, même si ces outils étaient interactifs.

Nous envisageons d'étendre les possibilités de notre application. Notamment de mettre à profit le modèle géométrique et le modèle multi-fenêtrage (qui sont complètement définis) pour spécifier l'aspect graphique d'une option menu (icône) qui, pour l'instant, est représentée par son nom dans un rectangle de taille fixe et identique pour toutes les options. Au passage, on peut noter qu'il s'agirait là d'un exemple de coexistence de deux applications (C.A.O et Conception Dialogue).

IV.2. UNE REPONSE AU PROGRAMMEUR

Ce point n'a pas encore fait l'objet d'une application interactive effective. Pour l'instant, toutes les actions sémantiques sont directement écrites dans un langage évolué (Pascal). Néanmoins, nous pouvons d'ores et déjà faire part de notre expérience dans ce cadre. Nous donnerons également des propositions qui orienteront nos choix pour l'écriture d'un générateur interactif d'applications.

Jusqu'ici, les seules personnes qui ont écrit des fonctionnalités sont des informaticien(ne)s membres de l'équipe. Nous les séparons en deux catégories :

- ceux qui sont "responsables" des fondements du modèle,
- ceux qui utilisent les bases du modèle telles qu'elles ont été définies.

Les premiers sont nécessairement moins objectifs que les seconds. Leur subjectivité peut être consciente, dans la mesure où ils entendent défendre leurs idées, ou inconsciente, car ils ne voient pas toujours les lacunes que d'autres relèveraient.

Toutefois, nous pensons avoir limité notre subjectivité car chacun a développé des aspects différents et fonctionnellement indépendants. Cette modularité a conduit à une diversité des problèmes soulevés et des solutions proposées. La synthèse de l'ensemble a justifié nos choix.

Par contre, dans la programmation proprement dite des traitements, il nous était difficile d'être complètement objectifs quant à la souplesse d'utilisation de la primitive INTERACTION. Aux dires de la deuxième catégorie de programmeurs, il semblerait que ce qui pouvait rendre l'utilisation d'INTERACTION un peu confuse était essentiellement dû au changement d'habitude qui en résultait. Une fois son principe compris, les différents programmeurs se rendait vite compte du travail dont ils étaient déchargés, d'une part, et de la

souplesse qu'introduisaient les diverses notions qui s'y rattachent, d'autre part. Notamment, la mise au point des traitements s'en est trouvée sensiblement accélérée.

Ceci a permis à des membres de l'équipe d'écrire des fonctionnalités nécessitant des dialogues complexes d'une façon naturelle et plutôt optimale. Par exemple, les PIECES PARAMETREES constituent une option qui combine un éditeur de texte et un éditeur graphique, le tout s'appuyant sur la primitive INTERACTION et sur les accès aux différents modèles.

Il faut souligner que le second groupe de programmeurs a apporté des points de vue nouveaux qui ont été pris en compte, mais qui n'ont pas nécessité de remise en cause des fondements de notre approche. Tout cela a permis de valider en partie certains aspects concernant la généralité et la facilité d'utilisation de nos principes.

Il est à présent indispensable d'aller jusqu'au bout de notre démarche, en écrivant une application interactive destinée aux non-informaticiens. Le but de cette application est de spécifier le plus simplement possible la sémantique profonde d'une action, et ce au travers d'un dialogue défini et présenté dans le cadre de notre modèle. Deux questions principales se posent :

- comment spécifier l'action,
- comment utiliser cette spécification.

Nous proposons des solutions à ces problèmes.

IV.2.1. SPECIFICATION DE LA SEMANTIQUE D'UNE ACTION

Cette fonctionnalité doit tenir compte de la caractérisation d'une action (cf. 5.2.2.). Dans la mesure où notre schéma général fait appel à des actions compilées, on devra pouvoir traduire leur description dans un langage de haut niveau afin de ne pas perdre les intérêts de ce choix, et afin d'éviter l'écriture d'un compilateur.

Bien que principalement destinée aux non-informaticiens, cette option doit être utilisable efficacement par des programmeurs confirmés. Elle doit donc prévoir la possibilité d'écrire directement du code dans le langage adopté. Ceci se résume à entrer dans un éditeur de texte. Cet éditeur doit pouvoir être, soit un éditeur existant que l'on chargerait par sélection d'un menu qui ferait un appel système, soit un éditeur écrit par nos soins avec les caractéristiques d'une action interactive. La première solution vise à permettre l'utilisation d'un éditeur auquel on peut être habitué. La seconde permettrait de profiter de tous les principes du système.

Pour satisfaire le non-informaticien, plusieurs possibilités s'offrent à nous :

- un langage de commande dont la syntaxe serait la plus simple et la plus synthétique possible. Par exemple :

  C = P , P , P

qui signifierait créer un cercle passant par trois points. Une telle phrase pourrait être par la suite automatiquement traduite en PASCAL :

```
  PORTEE(1,point);
REPEAT
      INTERACTION(1);
UNTIL NBOBJ(1) > 0;
COORD(OBJET(1),X1,Y1);
 .
 .
 .
 .
 .
COORD(OBJET(1),X3,Y3);
CREERCER3PTS(i,X1,Y1,X2,Y2,X3,Y3);
```

- un langage plus explicite, du type :

```
  P1 = INTERACTION;
P2 = INTERACTION;
P3 = 100,200 ;
C = P1 , P2 , P3 ;
```

- un diagramme d'états qui serait décrit graphiquement,

- une grammaire,

.....

Les phrases pourraient être introduites grâce à un éditeur ou par sélection de mots réservés. Quelle que soit la solution adoptée, elle nécessitera l'existence d'une bibliothèque de primitives (modélisations, affichages, calculs, ...) la plus fournie possible. Actuellement, notre bibliothèque n'autorise que la manipulation d'objets géométriques 2D. L'écriture d'applications dans d'autres domaines impliquerait des modules de base supplémentaires.

Concernant les applications graphiques, nous estimons qu'il est indispensable de fournir des principes génériques [GAR 87b] qui déchargeraient le programmeur de certains aspects. Par exemple :

- la façon d'afficher un objet étant connue, le déplacer continuement d'un point à un autre ne doit pas nécessiter de programmation spécifique. Autrement dit, une primitive DEPLACER(type) doit intégrer l'identification d'un objet de ce type, son réaffichage continu dans les positions intermédiaires et la validation de la position finale. On retrouve cette notion dans EZWin [LIE 85],

- pour tout objet dont la représentation graphique résulte d'un calcul sur N points, une primitive permettant un affichage élastique (rubber-banding) ELASTIQUE (calcul, N) doit intéger le fait de fixer les (N-1) premiers points, l'affichage de l'objet compte tenu de la position mobile du $N^{-ième}$ point jusqu'à validation de ce dernier .

Ce genre de primitives permettrait de spécifier très simplement des traitements interactifs apparemment complexes.

Nous pensons nous diriger vers un langage autorisant à la fois des phrases synthétiques, des phrases plus explicites, et la possibilité de sélectionner des mots clé. La raison principale de ce choix est due à l'existance de certaines fonctionnalités déjà prévues dans l'application C.A.O. Il s'agit des expressions graphonumériques dont le mécanisme permet d'alterner clavier et/ou souris pour construire une phrase qui est ensuite interprétée. De plus, une option PIECE PARAMETREES met en oeuvre l'interprétation d'une description qui répond à une syntaxe précise. L'intérêt est triple :

- il nous suffit d'étendre la syntaxe et l'interpréteur déjà existants,

- la même syntaxe serait utilisée partout, ce qui faciliterait l'apprentissage,

- comme son nom l'indique cette option gère la notion de paramètres, ce qui signifie que ses mécanismes vont nous permettre de paramétrer des actions. Notamment, le principe d'interprétation ne lance une INTERACTION que si le paramètre concerné n'est pas déjà fourni. De plus, l'interpréteur gère automatiquement les variables locales qui lui sont nécessaires. Autrement dit, nous avons bon espoir d'étendre le principe à une traduction automatique en procédure PASCAL.

Ce choix apporte également des éléments de réponse au second problème à résoudre.

IV.2.2. UTILISATION DE LA SPECIFICATION D'UNE ACTION

Un de nos principaux soucis est la possibilité d'utiliser immédiatement une action nouvellement spécifiée. L'intérêt étant d'éviter de sortir de l'application afin d'augmenter le dynamisme de l'association option-action.

Dans le cas d'une action interprétable, il suffirait de noter cet état de fait pour que la primitive APPEL lance son interprétation plutôt qu'une procédure déjà compilée. Ceci serait bien sûr moins performant (plus lent) mais permettrait une utilisation acceptable en cours de développement.

Lorsqu'une session de travail avec le générateur d'application se terminerait, toutes les actions spécifiées interprétables et définitives seraient traduites en PASCAL, compilées séparément et rendues exécutables par le primitive APPEL. Tout le processus serait automatique et ne nécessiterait aucune précision de la part de l'utilisateur (programmeur). Ainsi, à la prochaine session, les dites actions s'exécuteraient avec la rapidité qu'offre un code compilé.

Dans le cas d'une action directement écrite en PASCAL, son utilisation immédiate est plus délicate. Par contre, le processus de prise en compte automatique en fin de session serait le même que précédemment, exeptée la phase de traduction.

Lorsque nous avions décrit la primitive APPEL, nous avions défini une primitive SUIVANTCAS(action) qui, à partir du nom externe d'une action lance une procédure compilée interne. Le corps de celle-ci serait (en PASCAL) :

```
IF interprétable(action)
THEN
     INTERPRETER(action)
ELSE
     CASE action OF
        'nom1' : proc1;
          .
          .
        'nomn' : procn;
      END;
```

Dans le processus de prise en compte en fin de session, il suffirait de rajouter les cas manquants et de recompiler SUIVANTCAS avec les procédures concernées, le tout automatiquement.

Dans le cas d'une prise en compte immédiate d'un code compilé, il faudrait mettre en oeuvre des mécanismes plus complexes. La solution que nous proposons est la suivante. La primitive SUIVANTCAS prévoit à l'avance des appels à des procédures dont le code n'est pas encore défini, ou plus exactement dont le corps est vide. Par exemple :

```
CASE action OF
   'nom1' : proc1;
     .
   'nomn' : procn;

   'nomn+1' : procn+1;
     .
   'nomn+p' : procn+p;
END;
```

où les actions 'nom1' à 'nomn' sont celles déjà existantes et les actions 'nomn+1' à 'nomn+p' sont celles qui pourraient être définies. Les procédures procn+1 à procn+p ayant toutes le même corps:

```
PROCEDURE proci;
BEGIN
END;
```

et ayant été compilées séparément et éditées avec le programme principal. Ainsi, si une nouvelle action est écrite directement en PASCAL, ses noms externe et interne lui sont automatiquement attribués par le système, et pris parmi les couples (nomn+j,procn+j) avec 1<=j<=p. Si la première édition des liens s'est faite de façon dynamique, on peut momentanément sortir de l'application pour compiler le nouveau code et revenir automatiquement, ou appeler directement un compilateur si la place disponible le permet. Au prochain appel d'une telle procédure, son nouveau code objet sera automatiquement pris en compte sans refaire l'édition des liens.

Dans le cas où l'on sort pour compiler, il s'agit d'un DIFFERE au niveau système d'exploitation (niveau 0). Ce DIFFERE serait une suite de commandes systèmes qui se terminerait par un appel de l'application en cours (le générateur). Ainsi, on se retrouverait

automatiquement au début d'une nouvelle session en ayant l'impression de ne pas avoir quitté la session en cours. En fait, on est obligé de sortir pour relancer le programme principal si l'on veut que le nouveau code soit utilisable.

Il semble évident que le mécanisme peut être également appliqué à une action interprétable que l'on désire traduire immédiatement. De plus, le processeus de prise en compte en fin de session sera forcément simplifié dans la mesure où il ne sera pas nécessaire de modifier et recompiler la primitive SUIVANTCAS (puisque les nouveaux appels sont déjà prévus).

Nous venons de donner le principe général que nous prévoyons de mettre en oeuvre. Nous ne pouvons pas encore présager des difficultés que nous rencontrerons. Cela suppose l'existence d'un éditeur de liens dynamique. De plus, une compilation qui se passerait mal posera indubitablement des problèmes quant à la correction du source et au retour à l'application. Tout est à faire:

- élaboration d'un langage simple et efficace,
- interprétation d'une action,
- traduction automatique en PASCAL,
- caractérisation des éléments génériques,
.....

La spécification de la sémantique d'une application est certainement le point le plus difficile à réaliser. Il nous faudra affiner notre caractérisation d'une action afin de faciliter sa description interactive. Les expériences dans ce domaine sont, à notre connaissance, quasi-inexistantes [ROA 82]. La plupart des auteurs s'attache plus à la gestion et à la représentation externe du dialogue. Le sujet est vaste et très ouvert. Néanmoins, nous pensons que notre approche apportera des réponses acceptables, tout en ouvrant la porte à des solutions que nous n'avons pas envisagées. Des efforts importants doivent être faits dans ce sens car un générateur interactif d'applications est partie intégrante de notre modèle.

Ce paragraphe nous a permis d'entrevoir les solutions et de cerner les problèmes que nous aurons à résoudre. Nous sommes assez confiants car notre point de vue nous paraît cohérent et ouvert. A terme, notre intime conviction est que cela devrait conduire à un langage spécifié plus ou moins interactivement. Celui-ci devrait prévoir des structures de données et de contrôle prédéfinies en accord avec notre caractérisation d'une action et nos modèles (INTERACTION, menus, géométrie, multi-fenêtrage, ...). Nous aboutirions à un compilateur de langage graphique centré sur le concept d'INTERACTION unique et de tout ce qui s'y rattache. Le projet est ambitieux et constitue un sujet de recherche en soi.

CINQUIEME PARTIE

AUTRES MODELES ET AUTRES SYSTEMES

Nous exposons ce qui nous semble représentatif de l'état de la recherche en ce domaine. Nous nous attacherons à mettre en évidence quelles sont les différences et similitudes fondamentales par rapport à notre approche, plutôt que de les comparer les unes aux autres dans le détail.

Il y a presque autant de modèles différents que de points de vue différents. De plus la présentation des aspects externes d'un système ne permet pas toujours de se faire une idée juste des fondements de ses mécanismes. Malgré tout, on peut les regrouper suivant les grands courants de pensée dont se prévalent les auteurs eux-mêmes.

Ceux-ci se distinguent essentiellement par :

- le contrôle du flot d'entrées/sorties (interne, externe, les deux),
- le mode d'exécution (interprété, compilé, les deux),
- le degré d'indépendance par rapport au matériel,
- la capacité à redéfinir ou étendre ce qui est proposé par défaut,
- le type d'utilisateur privilégié (le concepteur, le programmeur, le commun des mortels, ou les trois),
- la nature et le nombre d'interactions possibles,
- des considérations d'ordre "philosophique".

V.1. LE MODELE LINGUISTIQUE

Cette approche est historiquement attachée à une longue tradition dans la formalisation des langages. PARNAS [PAR 69] et NEWMANN [NEW 68] sont certainement les initiateurs de l'utilisation de diagrammes d'états et de grammaires dans la spécification de systèmes interactifs.

Ils ont largement inspiré nombre d'autres approches. Qu'elles soient basées sur des grammaires plus ou moins complexes [HAN 80][MOR 81][REI 81], ou sur des diagrammes d'états et autres automates plus ou moins étendus [DEN 77][KAM 83][JAC 85][KIE 83][OLS 84b] ou récursifs [GUE 82][DEM 83], elles aboutissent toutes à des schémas comparables. Leur principe étant de générer du code à partir de la spécification qui sépare radicalement les niveaux lexical, syntaxique et sémantique.

Le niveau lexical précise la forme des entrées (sorties) possibles. Le niveau syntaxique précise les chemins suivis par l'analyseur (le déroulement du dialogue). Le niveau sémantique précise les actions intervenant à chaque étape de l'analyse syntaxique (entre autres, le code application à entreprendre).

Ce type d'approche conduit souvent à un contrôle exclusivement interne, dirigé par la syntaxe. En effet, en final, cela correspond à l'écriture explicite de toutes les directions que peut prendre le dialogue lorsqu'il se trouve dans un état donné. A savoir, toutes les règles de production qui définissent les dérivations possibles d'un non-terminal d'une grammaire, ou tous les arcs qui peuvent quitter un noeud d'un diagramme d'états.

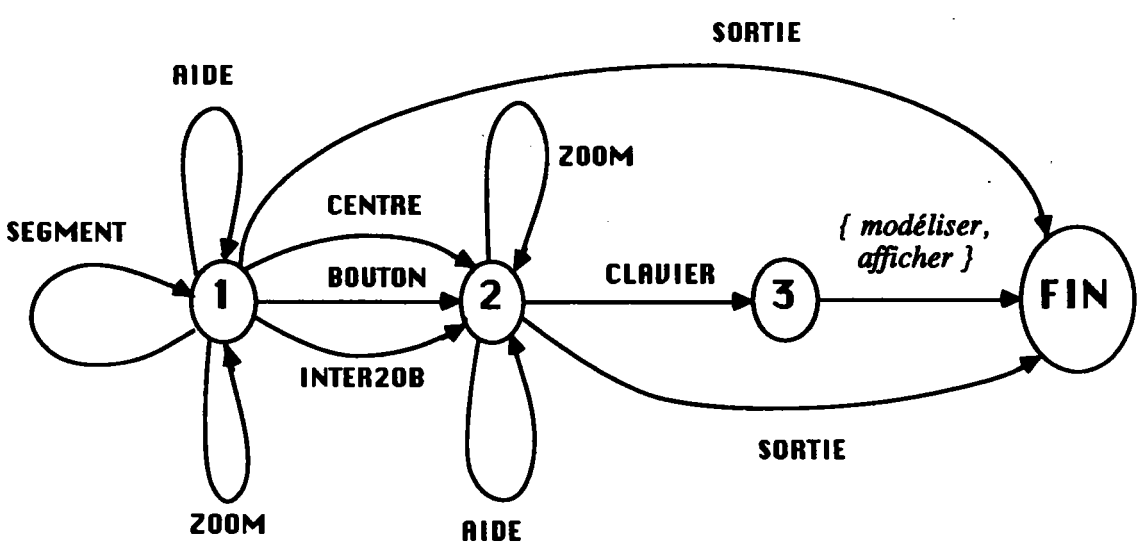
De plus, presque tous ces modèles considèrent de nombreux lexèmes (au niveau lexical) qui sont fonctionnellement l'équivalent des différentes primitives de dialogue que l'on retrouve dans d'autres approches (GKS, PHIGS, ...).

En revanche, le formalisme qui s'y rattache a l'avantage de bénéficier d'une longue expérience. Ce qui le rend assez fiable quant aux possibilités de prouver la consistance des interfaces qu'il permet de mettre en oeuvre de façon automatique.

D'aucuns [GRE 86] pensent que sa puissance de description est diminuée essentiellement du fait, qu'à l'origine, le modèle linguistique servait à décrire des systèmes (langages) non-interactifs. Notamment, les choix qui sont faits en cas d'impasse dans l'analyse (problèmes de non-déterminisme) ont tendance à handicaper le modèle.

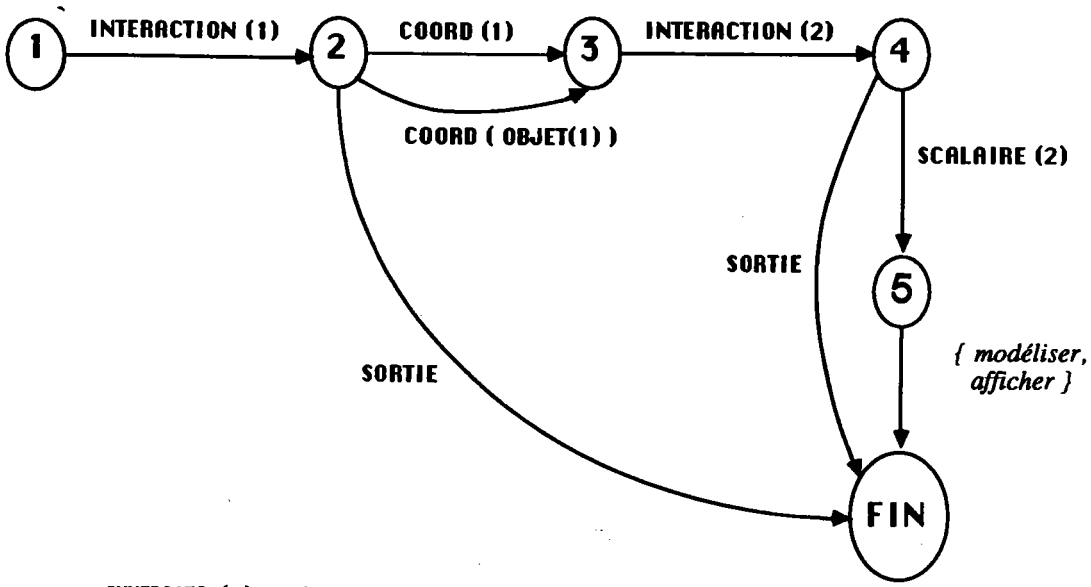
Comme d'autres [SIB 86], nous pensons que leurs limites ne sont pas tant dans le principe de leurs formalismes, mais plutôt dans l'utilisation qui en est faite. Dans notre approche, toute unité lexicale (menu, objet, clavier, souris, ...) est prise en compte par une unique unité syntaxique (INTERACTION) qui gère implicitement des notions sémantiques (compatibilité, localité, action associée, ...). Ainsi, un modèle linguistique peut être utilisé pour exprimer la sémantique profonde de l'application, sans pour cela remettre en cause les fondements de notre modèle. Ce constat était prévisible en raison de la dualité de notre approche : la partie contrôle interne entre, a priori, dans tout modèle où le contrôle est exclusivement interne. La différence essentielle est que certains appels (récursifs ou non) n'apparaissent pas explicitement dans le code.

Reprenons notre exemple du CERCLE. Un diagramme d'états dans le modèle linguistique donnerait :



(figure V.1.)

Alors qu'un diagramme d'états dans notre modèle serait :




```
  IMMEDIATS (1) = ZOOM + AIDE + SEGMENT
LOCAUX(1) = CENTRE + INTER2OB

IMMEDIATS(2) = ZOOM + AIDE
LOCAUX(2) = aucun
```

(figure V.2.)

Pour le même exemple, une grammaire dans le modèle linguistique pourrait être :

```
CERCLE   --> BOUTON | ZOOM | AIDE | SEGMENT
             | CENTRE | INTER2OB | SORTIR
BOUTON   --> {coord} RAYON
RAYON    --> CLAVIER | SORTIR | AIDE | ZOOM
CLAVIER  --> {valeur} FINIR
FINIR    --> {modéliser et afficher}
SORTIR   --> e
ZOOM     --> ZOOMER CERCLE
AIDE     --> AIDER CERCLE
SEGMENT  --> SEGMENTER CERCLE
INTER2OB --> INTERSECTER {coord point} RAYON
```

(figure V.3.)

Dans notre modèle, nous aurions :

```
CERCLE   --> INTERACTION1 COORD INTERACTION2 SCALAIRE
             FINIR | INTERACTION1 SORTIR
SORTIR   --> e
COORD    --> {coord} | {coord point} | SORTIR
SCALAIRE --> {valeur} | SORTIR
FINIR    --> {modéliser et afficher}

    IMMEDIATS (1) = ZOOM + AIDE + SEGMENT
    LOCAUX(1) = CENTRE + INTER2OB

    IMMEDIATS(2) = ZOOM + AIDE
    LOCAUX(2) = aucun
```

(figure V.4.)

On constate que l'on peut faire les mêmes remarques que celles que nous avions faites lors de la caractérisation d'une action (cf. III.1.2.). De telles utilisations des diagrammes ou des grammaires peuvent vite devenir confuses, dans la mesure où elles n'expriment pas uniquement la sémantique profonde d'une action. De même, ces approches limitent les possibilités de dynamisme (ne facilitent pas le prototypage, la maintenance, ...).

Par notre approche nous pouvons aisément utiliser un diagramme ou une grammaire pour exprimer l'essentiel d'une action (sa sémantique profonde) sans surcharger la notation. Nous pourrions également utiliser ces types de notation pour spécifier les Immédiats et les Locaux, par exemple, avec une grammaire:

```
IMMEDIATS1 --> Zoom + Aide + Segment
LOCAUX1    --> Centre + Inter2ob
```

Toute la différence réside dans la puissance du seul non-terminal INTERACTION dont les dérivations possibles sont implicites et prédéfinies, et qui est associé aux notions sémantiques que nous avons mises en évidence dans la troisième partie de notre exposé. De plus, ces notations pourraient servir à un formalisme plus rigoureux de la primitive INTERACTION elle-même.

En résumé, nous dirons que ce ne sont pas les formalismes eux-mêmes qui limitent la puissance de description, mais plutôt le fait qu'ils soient le point central des modèles d'interfaces qui en sont issus. Ces modèles sont surtout orientés programmeur, c'est-à-dire qu'ils rendent difficile la séparation des rôles.

Pour notre part, nous n'excluons pas leur utilisation en raison de la consistance qu'ils apportent dans la validation théorique de certains principes, d'une part, et dans l'automatisation de certaines mises en oeuvre, d'autre part. D'autant que sur le fond, notre modèle peut être vu comme un automate récursif dont le dynamisme autorise un "pseudo non-déterminisme". Nous pensons avoir introduit des concepts qui en facilitent la compréhension et qui permettent de dépasser les limites qu'imposent des utilisations "plus classiques" [KAM 83].

V.2. LE MODELE EVENEMENT

Un des soucis de ce modèle est d'uniformiser les notions d'entrées et de sorties par le concept d'événements asynchrones qui s'expriment indépendamment des unités physiques. Il considère toute entrée (sortie) comme un événement. Quand un événement se produit, il est envoyé à un (ou plusieurs) gestionnaire(s) d'événements.

Selon les auteurs [GRE 85][FLE 87][ANS 82][VAN 83][CAR 85], un gestionnaire est décrit suivant un formalisme plus ou moins différent; le principe étant de préciser pour chaque gestionnaire :

- ses paramètres,
- ses variables locales,
- les événements qu'il est capable de gérer,
- les procédures qu'il utilise.

La description se présente souvent de façon procédurale. En fait, les paramètres d'un gestionnaire représentent des unités lexicales associées aux événements qu'il peut gérer.

Contrairement au modèle précédent, le contrôle n'est pas uniquement dirigé par la syntaxe. En effet, en plus de sa nature, la valeur d'un lexème peut être utilisée, ce qui permet d'introduire une notion sémantique dans le contrôle. En fait, les diagrammes augmentés [DEN 77][KAM 83][JAC 85] offrent des possibilités comparables.

La description d'un gestionnaire est analogue à ce que nous avons défini pour une action, elle donne le comportement d'une option. Il s'agit d'une énumération d'événements associés à des traitements à effectuer si un événement se produit. L'ordre dans lequel apparaissent les événement n'a pas d'importance. C'est ce qui donne l'illusion d'asynchronisme du contrôle.

Un événement d'entrée est analogue à nos primitives d'accès aux résultats d'une INTERACTION.

L'ensemble des gestionnaires constitue l'interface. Ils sont traduits en procédures dans un langage évolué (souvent en C) et associés à des structures qui permettent à un interpréteur de dialogue de gérer le tout à l'exécution, par réceptions et envois d'événements. En fait, les gestionnaires sont des descriptions génériques. A l'exécution, **ils sont instanciés.**

Si un événement est produit par une unité d'entrée, il est envoyé à toutes les instances actives qui l'acceptent. Si un événement est explicitement produit par le gestionnaire de dialogue, l'instance destinataire est précisée.

Voici ce que pourrait être notre exemple dans un tel modèle:

```
GESTIONNAIRE  CERCLE

{
 Lexèmes acceptés : Bouton, Valeur, Centrecercle,
                    Inter2ob ;
 Variables locales : centre, rayon, etat, zoomi,
                     c, aidei, segmenti, centrei ;

 Evénément  INIT  /* automatique à l'instanciation */
  {
    zoomi = Instancier(ZOOM) ;
    aidei = Instancier(AIDE) ;
    segmenti = Instancier(SEGMENT) ;
    Envoyer(aidei,Message,"donnez ou créez un point") ;
    etat = 0 ;
  }


 Evénément  Bouton (couple)
  {
    SI etat=0
      centre = couple ;
      Envoyer(aidei,Message,"donnez un réel") ;
      etat = 1 ;
      Détruire(segmenti) ;
    FSI
  }
```

```
    Evénement  Valeur (r)
     {
       SI etat=1
         rayon=r ;
         Détruire(zoomi) ;
         Détruire(aidei) ;
         créercercle(c,centre,rayon) ;
         affichercercle(c) ;
       FSI
     }


    Evénement Centrecercle
     {
       SI etat=0
         centrei=Instancier(CENTRE) ;
         Envoyer(centrei, se faire) ;
         etat=1 ;
         centre=coord du point créé ;
         Détruire(centrei) ;
       FSI
     }


    Evénement  Inter2ob
     {
       SI etat=0
         centrei=Instancier(INTER2OB) ;
         Envoyer(centrei, se faire) ;
         etat=1 ;
         centre=coord du point créé ;
         Détruire(centrei) ;
       FSI
     }

  } /* fin du Gestionnaire CERCLE */



GESTIONNAIRE  AIDE

 {
   Lexèmes acceptés : Aider, Message ;
   Variables locales : messi ;

   Evénement  Message (mess)
    {
      messi=mess ;
    }


   Evénement  Aider
    {
      afficher(messi) ;
    }

 } /* fin du Gestionnaire AIDE */
```

La notation que nous utilisons ne répond à aucune règle précise que l'on pourrait trouver dans les différents modèles de ce type. Nous la prenons comme support de discussion.

Les gestionnaires compatibles n'apparaissent pas dans la liste des lexèmes acceptés, on déclare des variables locales qui référenceront leurs instances.

La spécification de CERCLE doit explicitement instancier (activer) les gestionnaires qu'elle admet comme compatibles. Ainsi, si un événement accepté par l'un d'eux se produit, il est automatiquement pris en compte. A la suite de quoi, on se retrouvera dans CERCLE comme si rien ne s'était passé. Il s'agit d'un empilement équivalent à celui de nos Immédiats.

Ceci n'empêche pas CERCLE d'envoyer des événements à des instances de certains d'entre eux. Par exemple, Envoyer(aidei,Message,".....") initialise le message d'AIDE qui ne s'affichera que si une AIDE est demandée (événement Aider reçu).

La notion de variables locales est analogue à la nôtre car elle permet de se décharger de la sauvegarde de l'environnement.

Les gestionnaires CENTRE et INTER2OB ne peuvent être utilisés comme les autres, sans quoi ils seraient empilés de la même façon et leurs résultats ne pourraient être utilisés puisqu'on reviendrait dans CERCLE comme si rien ne s'était passé. Ceci explique la nécessité de prévoir deux lexèmes, Centrecercle et Inter2ob, afin de lancer explicitement les deux gestionnaires associés. Cette déclaration montre leur caractère local.

CERCLE doit explicitement détruire (déactiver) les instances qui ont été activées.

Il n'y a pas de séquencement explicite afin de permettre l'asynchronisme des événements. Mais ce n'est qu'une illusion car la variable "etat" est inévitable pour un comportement cohérent de l'ensemble.

Dans la mesure où tout événement est automatiquement envoyé à toutes les instances actives et que l'une d'elles n'exécutera que les actions associées à l'événement reçu, le contrôle est externe. Néanmoins, l'utilisation de variables d'état locales, ainsi que la nécessité d'instancier (ou de détruire) des gestionnaires explicitement dans les actions associées à des événements, rendent le contrôle "indirectement" interne.

Il nous semble que c'est ce qui pourrait expliquer que la description des gestionnaires peut vite devenir confuse. Notamment, l'aspect très local et trop explicite du concept de compatibilité lui confère un caractère statique. En effet, le comportement du code résultant de la compilation de ces descriptions ne pourra être modifié qu'en changeant et en recompilant les descriptions.

En fait, contrairement à notre approche, un événement lié à une option menu n'a rien de particulier; il est considéré (et traité) comme tout autre événement d'entrée ou de sortie. L'uniformité de cette approche nous paraît excessive dans la mesure où elle ne répond pas de façon satisfaisante à nos propres objectifs de dynamisme.

En revanche, il est clair qu'elle assure l'indépendance par rapport aux unités physiques. De plus, elle trouvera toute sa signification dans une architecture multi-processeurs autorisant de réels accès concurrentiels. Encore qu'au vu de notre expérience, le

fait de prendre en compte des événements indépendamment du contexte d'utilisation ne paraît pas très réaliste. Cela oblige à une spécification explicite de mécanismes qui, selon nous, devraient être transparents.

Concernant nos préoccupations immédiates, de la même façon que pour le modèle linguistique, nous n'excluons pas complètement cette approche car :

- elle est indépendante du matériel,

- il y a une analogie entre événements d'entrées et primitives d'accès aux résultats d'INTERACTION,

-il y a une analogie entre envois d'événements et primitives d'initialisation des champs d'INTERACTION ou primitives d'accès aux différents modèles (géométriques, affichages, ...).

On trouvera une étude comparative du modèle linguistique et du modèle événement dans [GRE 86]. Ces modèles ont le mérite d'exister et d'être entièrement formalisés. Mais nous sommes convaincus que ces approches se limitent d'elles-mêmes par leur tendance à fournir des outils trop généraux pour définir des interfaces. Ces outils sont plutôt adaptés au programmeur d'application. En fait, il y a désaccord quant à la place qu'occupent les divers composants (ou en tout cas sur la terminologie employée). Par exemple, dans le modèle événement, les gestionnaires définissent l'interface, et les traitements associés aux événements définissent l'application. Alors que pour nous, ces gestionnaires seraient des actions (au sens où nous l'avons défini) et seraient partie intégrante de la sémantique de l'application interactive. L'interface proprement dite est pour nous l'ensemble des mécanismes implicites et des notions générales (standard) que nous avons introduit.

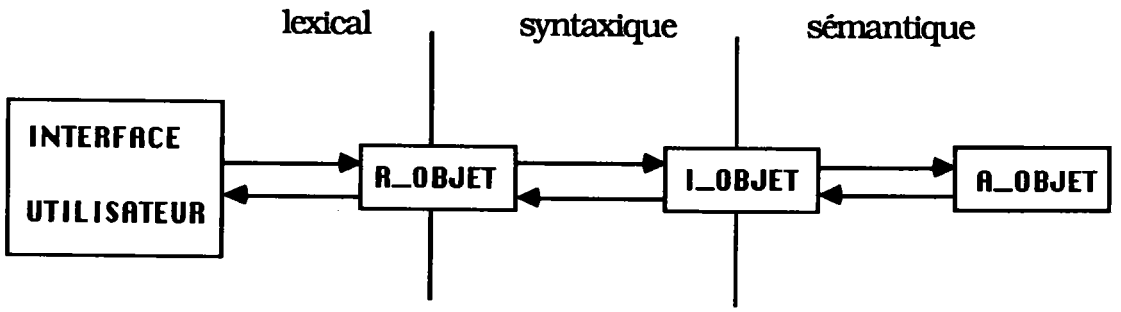
V.3. MODELES ORIGINAUX

Les modèles que nous venons de voir sont très généraux dans la mesure où ils n'ont pas d'orientations particulières. Ils permettent aussi bien de spécifier un système orienté menus qu'un système orienté langage de commande (ou autre). En conséquence, ils n'imposent pas une technique d'interaction plus qu'une autre. Ils ne fixent pas des aspects "standard" d'une application interactive.

D'autres modèles, moins bien formalisés, cherchent à mettre à profit un point de vue particulier afin d'éviter d'avoir à spécifier explicitement des mécanismes de "bas niveau" considérés communs à la majeure partie des applications. Nous relatons ici quelques expériences qui ont suivi ce type de démarche.

Les deux premiers exemples sont des modèles orientés objets, au sens où ils utilisent des notions de la programmation objet non pas uniquement pour l'implantation, mais surtout pour les concepts fondamentaux qui les caractérisent.

Le système GWUIMS [SIB 86] est défini à partir d'objets de représentation, d'objets d'interaction, d'objets d'application et d'une interface utilisateur. Ces composants sont liés suivant le schéma :



(figure V.5.)

L'interface regroupe l'ensemble des techniques d'interaction (souris, clavier, ...) associées à des unités d'entrée, ainsi que les unités de sortie et tout ce qui est susceptible d'être affiché. Comme le montre la figure, les R_objets assurent un lien entre le niveau lexical et le niveau syntaxique, les I_objets un lien entre les niveaux syntaxique et sémantique. Les A_objets correspondent à la sémantique de l'application. Les différents éléments communiquent entre eux par envois de messages.

Hormis les ponts qui existent entre les niveaux, le système part du principe qu'il y a une différence entre sélectionner une entité, entrer une valeur et renvoyer un résultat. C'est pourquoi, un I_objet communique avec les autres par sélection ou par initialisation d'attributs typés qui lui sont associés. Ces attributs peuvent être actifs ou passifs. Il en existe trois sortes :

- un article ("item") est initialisé à la conception et ne peut être modifié à l'utilisation (ex. un menu),
- une case ("slot") que l'utilisateur peut modifier par une entrée,
- un résultat que l'application peut modifier à l'exécution.

Lorsqu'un attribut actif est accédé, un message est envoyé à un A_objet (associé au I_objet dont dépend l'attribut) qui exécute une action.

Un I_objet communique les entrées provenant d'un R_objet à un A_objet et l'action associée lui renvoie ses résultats qui sont retournés à un R_objet (pas forcément le même qu'en entrée).

Plus que la façon dont circulent les informations, ce qui nous semble intéressant dans cette approche c'est le rôle central des I_objets. En effet, ils ont connaissance de la syntaxe et de certains aspects sémantiques. En d'autres termes, à ce niveau, le concepteur d'interface peut spécifier des éléments du dialogue, ce qui

normalement incomberait au programmeur. De plus, en modifiant des attributs résultats, l'application peut avoir une influence sur le contrôle du flôt.

A ce titre, on trouve une certaine analogie entre les I_objets et l'INTERACTION dans notre modèle. Il y a également une similitude entre l'invocation automatique d'une action lors de l'accès à un attribut actif d'un I_objet et l'APPEL d'une action. Les R_objets sont analogues à la partie "interprétation d'une intention" en entrée et aux variations d'aspect visuel en sortie. Ici, ils sont redéfinissables à volonté, alors que chez nous ils sont figés.

Les auteurs ne donnent pas d'autres exemples que la relation qui existe entre la représentation externe d'une commande et l'action associée. Il est donc difficile d'entrevoir ce que donnerait une application effective sans exemples plus détaillés des A_objets. Néanmoins, il semblerait que l'association commande-action soit dynamique et permette d'éviter certaines recompilations.

Les sorties plus spécifiques d'une application, comme l'affichage d'un objet géométrique par exemple, devraient suivre le schéma ci-dessus. Si c'est bien le cas, il est clair qu'un I_objet est à prendre dans sa généralité, c'est-à-dire comme objet de communication au sens large, plutôt qu'interaction au sens où nous l'entendons dans notre modèle.

Il nous semble intéressant de citer également l'expérience menée par LIEBERMAN [LIE 85]. Celui-ci propose le système EZWin, basé sur d'autres classes d'objets. EZWin est orienté menu et ses composants sont des objets de trois types possibles :

- un objet de classe EZWin qui représente l'interface lui-même,
- des objets de classe Représentation (tout ce qui susceptible d'être visualisé dans une application),
- des objets de classe Commande (toute action [option] dont un utilisateur peut demander l'exécution au système).

L'objet EZWin est unique. Il est l'équivalent de notre NOYAU. C'est la boucle d'interaction définie une fois pour toutes quelle que soit l'application. De ce point de vue, cet objet se veut de répondre aux mêmes soucis de généralité et de consistance qui ont justifié nos propres choix.

Un objet Représentation spécifie comment un objet graphique propre à l'application doit être affiché.

Un objet Commande spécifie son aspect visuel, l'action qui lui est associée, ainsi que le type de ses paramètres. C'est l'analogue des menus de notre structure.

Une différence avec notre approche est la volonté de permettre indifféremment des commandes post-fixées, pré-fixées ou infixées, alors que dans notre modèle, toutes les options sont pré-fixées. Ceci explique la présence des types des paramètres d'une action associée à une commande.

En fait, la boucle principale récupère tout objet ayant un sens dans le contexte courant, que ce soit une commande ou un objet de l'application. Si c'est une commande qui est sélectionnée, tout objet dont le type n'est pas parmi ceux précisés dans la liste des paramètres sera automatiquement ignoré à la prochaine identification. Si c'est un objet Représentation, toutes les commandes susceptibles de travailler avec un premier argument de ce type sont mises en évidence (ou apparaissent dynamiquement). Si un second objet Représentation est identifié, l'ensemble des commandes en évidence changera en conséquence.

Lorsqu'une commande dispose de tous ses paramètres, l'action associée s'exécute automatiquement. Dans le cas d'une commande dont le nombre de paramètres est variable, une commande d'exécution doit être explicitement sélectionnée.

De plus, le système gère une liste des objets Représentation créés et détruits. Chaque fois qu'un tel objet est créé (respectivement détruit), il est automatiquement affiché (respectivement effacé). Si bien qu'en définitive, la boucle principale se comporte de façon analogue à la boucle "read-eval-print" d'un interpréteur LISP (par exemple). Sur ce plan, notre approche est fondamentalement différente, car nous considérons que la modélisation et l'affichage doivent être séparés. Il faut pouvoir modéliser sans nécessairement afficher. L'affichage n'est qu'une utilisation particulière du modèle, au même titre qu'un calcul de surface, ou de programme de commandes d'usinage.

Tout objet susceptible d'être sélectionné à un instant donné est mis en surbrillance chaque fois que le curseur passe dessus. L'auteur appelle cela: "sensible à la souris". L'idée est comparable à la notion de PORTEE que nous avons définie, excepté qu'ici, le test de compatibilité de type est fait en permanence pendant le déplacement de la souris (ce qui serait difficilement acceptable pour une application C.A.O., compte tenu de la diversité et de la complexité des objets manipulés).

L'auteur justifie la liste des types de paramètres par le fait que le code d'une action doit pouvoir s'utiliser en dehors de cet environnement. Autrement dit, les paramètres d'une action peuvent être fournis par un programme utilisateur. Le principe est intéressant, mais il nous paraît risqué, car le type des arguments peut ne pas suffir. En effet, il est des traitements qui manipulent des objets de même type, mais dans un ordre précis. De plus, le type d'objet attendu peut être dépendant du déroulement de l'action elle-même. Les notions de conditionnelles et d'itérations risquent d'être difficiles à prendre en compte dans ce modèle.

Un autre point à remarquer dans cette approche est la possibilité d'inclure la façon de créer un objet graphique dans son type. Par exemple, une commande peut attendre un objet de type "rectangle1", avec la définition de ce type qui précise qu'il est obtenu par deux coins diagonalement opposés. Si bien que si l'on change le type "rectangle1" par "rectangle2", on pourra spécifier la même commande avec un rectangle défini par son centre, sa largeur et sa longueur. Le dialogue de construction du rectangle est défini par son type "rectangle1" ou "rectangle2". Le type de l'argument peut très bien ne

rien préciser, ce qui signifie que le moment venu, le paramètre sera sélectionné parmi les objets "sensibles à la souris".

Le mécanisme est comparable à notre utilisation des Locaux qui permet de créer un objet plutôt que de l'identifier. Néanmoins, ce n'est pas le même concept, puisque pour nous, il s'agit de sélectionner un menu pour définir un objet, alors qu'ici il s'agit d'un aspect intrinsèque à la définition de l'objet lui-même. Par ailleurs, notre approche autorise plusieurs constructions possibles à un instant donné en spécifiant plusieurs Locaux, alors qu'ici un argument n'a qu'un seul type, donc un seul mode de construction. Enfin, ici, et pour les mêmes raisons, si un paramètre peut être identifié, il ne peut être construit.

Les deux exemples que nous venons de citer s'appuyaient sur des concepts objets (au sens SMALLTALK....). Ces choix étaient motivés par l'uniformité de la notion de message, d'une part, et par les mécanismes d'héritage des propriétés que sous-entendent ces modèles, d'autre part. Ainsi, des classes d'objets ayant été prédéfinies avec leurs propriétés et leur méthodes (procédures), elles peuvent être instanciées pour définir des interfaces qui bénéficient automatiquement de la transparence de ce type d'environnement. Il faut noter qu'en général, cela nécessite un interpréteur (type SMALLTALK). Ce qui implique une perte de place et de temps.

Le plus important à retenir, ce sont leurs principes, plus que leurs implantations. Ces modèles tendent à "standardiser" les techniques d'interaction, et, par là-même, la vue externe de l'interface. On remarquera que nombre des aspects de notre modèle peuvent s'exprimer en terme de classes d'objets, de méthodes et d'héritage.....

Notons également que le modèle Evénement s'inspire en partie des approches objets. Ses Gestionnaires sont comparables à des classes d'objets que l'on instancie. Ses Evénéments se comportent comme des messages dont la réception provoque des actions (méthodes). Il se différencie par l'inexistence d'héritage, et par l'asynchronisme des Evénements qui s'oppose au synchronisme des messages.

Continuons notre tour d'horizon avec le système développé et utilisé par le constructeur d'automobiles FIAT [ROM 87]. Ce système travaille bien entendu dans un cadre très spécifique. Néanmoins, on y retrouve des concepts très généraux. Il est orienté menus, avec une évaluation pré-fixée des commandes.

A tout moment, l'ensemble des options est accessible et l'état du système est clairement visualisé. Il peut y avoir une arborescence, mais d'une manière générale, c'est un système à un niveau. Le principe est toujours le même :

- choix d'une option,
- choix des paramètres,
- résultats.

Au moment d'identifier un argument, l'utilisateur peut :

- le montrer ou le nommer,
- le construire,
- modifier l'affichage (pour mieux choisir par exemple).

La réentrance possible des commandes conduit à un empilement des contextes.

Si une option est ajoutée au système, elle est globale, et à ce titre, accessible à tout moment.

On constate donc que ce système est très similaire au nôtre, avec toutefois quelques différences.

Ici, toute option est compatible avec une autre, et ce de façon définitive, alors que dans notre système, toute option est par défaut incompatible avec une autre, mais surtout la compatibilité est reconfigurable à volonté.

Dans ce système, le fait de sélectionner un menu de création, au lieu d'identifier directement, est apparemment considéré au même titre que le fait de changer l'affichage (en effectuant un "zoom", par exemple). Nous ne comprenons pas très bien comment les deux mécanismes peuvent être unifiés, dans la mesure où le premier fournit le résultat de l'identification, et que le second ne restitue rien.

De la même façon, les auteurs ne précisent pas ce qui se passe si, par exemple, on autorise la création d'un cercle alors que le paramètre attendu est de type segment. Pour que cela se passe bien, cela implique une exclusion implicite des options qui ne conviennent pas, donc un moyen d'en décider.

De plus, dans le même ordre d'idées, il nous est difficile de savoir ce qu'il en est des options qui ne sont ni des créations, ni des modifications de l'affichage.

Compte tenu de l'usage qui en est fait, il semblerait que ce système se soit volontairement limité aux aspects évoqués ci-dessus. Le petit nombre de principes à retenir, et le relatif manque de dynamisme, sont, dans ce cadre, des avantages. En effet, le système n'en est que plus facile à utiliser et à apprendre. Pour notre part, nous le considérons comme exemplaire dans les différents compromis qu'il réalise, étant bien entendu qu'il n'atteint pas tous les objectifs que nous nous sommes fixés.

Le système TIGER [KAS 82] présente de grandes similitudes avec le nôtre. Un interpréteur parcourt un arbre issu de la compilation d'une description par langage de commande et de contrôle (TICCL). Cet arbre est comparable par son rôle à notre structure de menus.

L'aspect externe de l'interface est le même pour toute application. Les notions de compatibilité s'y retrouvent de manière analogue. A savoir, toute commande est implicitement incompatible avec une autre, à moins d'en décider autrement. En cas de sortie due à un menu incompatible, le programmeur doit également en tenir compte.

On peut relever des différences. TIGER propose un contrôle presqu'exclusivement externe, dans la mesure où, tout comme EZWin, il gère également les paramètres des commandes. Ce qui permet de ne pas imposer l'ordre commande-arguments.

De plus, un principe de chemin (commande) par défaut autorise un ordre quelconque des paramètres eux-mêmes. Ceci donne la possibilité de revenir sur un premier argument alors que plusieurs ont déjà été précisés, ou de réappliquer un traitement, dont les paramètres ont déjà été initialisés, à un nouvel argument. Un chemin par défaut peut être spécifié explicitement dans la description, ou déduit implicitement par des règles établies. En fait, le dialogue est défini par des niveaux analogues à nos niveaux de Locaux (CONTEXTEs). Schématiquement, un chemin par défaut est celui que l'on peut suivre en descendant dans la hiérarchie des niveaux. Plusieurs commandes peuvent se trouver à un même niveau. L'une d'elles est celle par défaut.

Ce principe n'est pas prévu dans notre modèle, mais nous pourrions obtenir le même type de fonctionnalités par une utilisation judicieuse des Locaux, d'une part, et par la réitération automatique d'un traitement, d'autre part. En fait, en raison du contrôle externe portant sur la saisie d'arguments, la description englobe à la fois l'organisation des commandes et leur déroulement. Ce qui signifie que tout est interprété syntaxiquement, y compris le déroulement d'une action. En ce sens, les deux approches sont différentes, puisqu'à ce niveau, notre schéma est compilé.

Par ailleurs, il semblerait qu'en dehors de la structure de niveaux, le concept de Locaux, au sens où nous l'entendons, n'existe pas dans TIGER, notamment en ce qui concerne leur incidence dans le déroulement interne d'une action, et leur corrélation avec la désignation d'un objet.

Le principe de TIGER montre qu'il n'y a pas d'association dynamique option-action. Modifier le comportement d'une commande, c'est changer et recompiler sa description. En définitive, nous aurions tendance à penser qu'il s'agit d'un contrôle externe, vu de la sémantique d'une action, mais qui devient totalement interne, vu de l'interpréteur. En d'autres termes, c'est une utilisation subtile du modèle linguistique, à laquelle s'oppose radicalement la dualité de notre approche. Ceci explique, en particulier, que le programmeur est ici responsable de la sauvegarde des variables locales d'une fonction. L'interpréteur se charge de la sauvegarde et de la restauration des contextes, de manière analogue à notre empilement de CONTEXTEs.

Toutefois, la démarche suivie par ce système est, dans son ensemble, assez comparable à la nôtre, et conduit à des notions très similaires de "standardisation" de l'interface qui visent à renforcer sa consistance.

Nous terminerons par le système ADM [SCH 85]. Ce système est l'un de ceux qui accordent un rôle dual au gestionnaire de dialogue et à l'application. Sans entrer dans le détail, disons simplement qu'il s'appuie sur la compilation d'une description déclarative du dialogue (aspects visuels, hiérarchie des commandes, ...) et sur la notion de

tâche qui provoque un événement à partir duquel l'application dirige son déroulement.

L'application est écrite en langage évolué (Pascal, C, Fortran, ...) et peut utiliser des primitives d'accès à une tâche. Conceptuellement, une tâche est l'analogue de notre INTERACTION. ADM considère qu'il n'y a que deux types d'événements :

- pointer à une position écran,
- appuyer sur une touche ou un bouton.

Un événement bouton est associé à une série de couples (représentation externe, action à exécuter sur la représentation externe). La représentation externe peut être nommée explicitement ou être celle qui se trouve sous le curseur. Ceci donne le lien entre une unité lexicale et sa prise en compte syntaxique. Les actions peuvent être :

- sélectionner une option,
- donner une aide concernant une entité,
- .....

Un événement peut ainsi provoquer des appels à des routines application, ou des changements d'état du système, ou encore le passage du contrôle du gestionnaire à l'application (ou inversement).

Bien que nous pensons que notre système offre plus de possibilités de par certains aspects fondamentaux que l'on ne retrouve pas dans ADM, la base des deux systèmes est la même.

Dans ce paragraphe, nous nous sommes attachés aux principes de certains modèles, plus qu'à leurs implantations. Il nous a semblé difficile de faire la part des choses, d'autant que tous proposent des fonctionnalités attrayantes. Au regard de toutes ces expériences, nous sentons que notre approche tend à en être une synthèse, ceci étant une conséquence et non un but en soi. C'est pourquoi nous sommes confortés dans nos idées de départ.

CONCLUSION

Notre exposé se voulait le plus général et le plus fonctionnel possible afin de mettre en évidence l'essentiel des concepts que nous entendons défendre.

Nous nous sommes plus particulièrement attachés à la gestion du dialogue que nous avons située dans une architecture logicielle d'ensemble. Nous avons essayé de montrer comment une approche duale pouvait répondre aux préoccupations d'un opérateur, d'un concepteur d'interface et d'un programmeur.

Dans ce cadre, il nous semble que nous avons introduit des concepts nouveaux qui se caractérisent par leur généralité et par la puissance qu'ils procurent dans la description d'interfaces conviviaux.

Paradoxalement, l'uniformité de notre point de vue le rend ouvert. En effet, nous avons tenté de localiser et de caractériser les composants clé du dialogue. Leurs places et rôles dans notre modèle lui confèrent un aspect "standard". Cela étant, le dynamisme de nos principes autorise des extensions sans pour autant remettre en cause leurs fondements. Par exemple, les attributs d'une INTERACTION (les champs de la structure) que nous avons rencontrés étaient un support de discussion. Ce sont ceux que nous avons effectivement implantés. Les modifier ou les étendre ne changerait rien au principe d'INTERACTION unique. Entre autres, nous envisageons d'ajouter d'autres IMMEDIATS prédéfinis, comme par exemple, une option UNDO pour défaire ce qui vient d'être fait. De même, nous sommes en train d'implanter la sauvegarde automatique d'une session de travail. Cela sera grandement facilité du fait que tout est centralisé dans l'INTERACTION. L'utilisation de nouvelles unités physiques d'entrée ne nécessitera que de "petites" modifications de bas niveau dans l'interprétation d'une intention.

La démarche que nous avons suivie nous parait cohérente. Nous pensons avoir facilité la compréhension et l'apprentissage de notions complexes en les intégrant dans des concepts relativement simples.

Nous n'avons pas formalisé notre approche de façon rigoureuse. Néanmoins, il faut noter que les différents formalismes rencontrés visaient essentiellement à décrire les outils mis à la disposition du concepteur. De ce point de vue, notre générateur interactif de dialogue rend inutile le développement de tels formalismes. Il en va de même concernant le générateur d'application. Par contre, il ne fait aucun doute qu'il nous faudra décrire plus rigoureusement notre modèle. Toutefois, les structures et mécanismes qui le composent peuvent être considérés en eux-mêmes comme une formalisation. Par ailleurs, nous avons pu remarquer que nombre des différentes approches existantes peuvent être en partie utilisées dans ce cadre. Autrement dit, nous aurions tendance à penser que la défense de tel ou tel formalisme est un faux débat. Plus précisément, la façon de décrire un système n'est pas toujours une question de fond. Par exemple, dire qu'un menu est en partie défini par une action qui s'exécute quand on le sélectionne, au lieu de dire qu'un menu est un objet dont la méthode associée se déclenche à la réception d'un message de sélection, ou encore l'entrée d'une unité lexicale fera quitter un état (noeud) suivant un arc qui nous conduira à la prise en compte

d'une action sémantique, sont autant de façons de dire (décrire) la même chose. En ce sens, cela ne nous semble pas fondamentalement important.

Le caractère très ambitieux de notre projet nous laisse encore énormément de travail. Les développements d'une application C.A.O. 2D et d'une application orientée concepteur sont des étapes qui nous ont permis de valider la plupart de nos principes. Forts de ces acquis, nous entendons poursuivre dans le même sens afin de consolider l'ensemble par une application interactive orientée programmeur, pour laquelle nous avons entrevu des solutions. Ici aussi, nous envisageons de profiter de l'unicité de la primitive INTERACTION, associée à des notions génériques, pour simplifier l'outil.

Les limites matérielles qu'impose un environnement microordinateur n'ont que très peu été abordées. Notre première implantation du système semble acceptable d'un point de vue expérimental. Elle devra être remise en cause dans un cadre plus étendu (augmentation de la taille du code et des différents modèles).

Enfin, il est clair que la couche 3D doit être intégrée. Actuellement, c'est le cas concernant le modèle générique et certains modèles de visualisation. En revanche, le modèle multi-fenêtrage doit être étendu. Nous envisageons de permettre l'association d'une fenêtre à un plan de projection quelconque. Ainsi, chaque fenêtre serait une vue 2D du modèle 3D, et un modèle générique du type de celui vu en II.3. pourrait lui être attaché. La plupart des possibiltés du modèle de dialogue et du modèle de visualisation 2D resteraient les mêmes. Il suffira de prévoir les processus manquants de globalisation (désignation 3D, coordonnées 3D, ...) et de localisation (projections diverses). La cohérence qui existe déjà entre les différentes vues (fenêtres) d'une scène 2D permettra la cohérence des différentes projections d'un modèle 3D.

ANNEXE 1

Dans la première partie, nous avions présenté certains aspects externes de SACADO 2D (conventions, ...). Au travers de quelques fonctionnalités (options), cette annexe est l'occasion de donner des exemples de dialogues qui visent à montrer les apports de notre approche en matière de convivialité. Notamment, ces exemples soulignent les intérêts de la primitive unique "INTERACTION", et des notions d'"IMMEDIATS" et de "LOCAUX".

De plus, il nous paraît important de citer des solutions adoptées dans le but d'augmenter la souplesse d'utilisation (détection et proposition automatiques des choix possibles, diminution du nombre d'interactions, ...).

Nous terminons par un exemple de plan qui donne une idée des possibilités de SACADO 2D (réalisme de l'application).

I. EXEMPLES

I.1. Les constructions sous contraintes

Une construction sous contraintes est définie par:

- le type d'objet à construire (point, segment, arc, cercle),

- des contraintes principales (tangent à, passant par, distant de...),

- les objets sur lesquels portent les contraintes principales,

- des contraintes secondaires permettant de choisir un objet parmi plusieurs (par exemple, un cercle tangent à deux cercles de rayon donné, peut être choisi parmi au plus 8 cercles candidats).

L'objectif a été de prendre en compte ces contraintes en utilisant les menus dynamiques et en minimisant le nombre de contraintes secondaires à préciser, ce qui rend le dialogue beaucoup plus simple que dans les solutions traditionnelles.

Le dialogue se déroule de la manière suivante:

- l'opérateur choisit par menu le type d'objet à construire,

- le sous-menu des constructions principales associé au type d'objet choisi apparaît, et l'opérateur peut choisir l'une des constructions,

- en fonction de la construction demandée, le système demande des identifications ou des coordonnées,

- à partir des renseignements donnés, le système calcule les solutions répondant aux contraintes principales, recherche la solution la plus plausible, qui devient la solution courante et que nous appelons pour l'explication OBJET (voir exemple ci-dessous), et la propose (l'affiche). En situation de PROPOSITION DE SOLUTION, les cas suivants peuvent se produire:

- si l'utilisateur identifie un nouvel objet(ou donne de nouvelles coordonnées, en fonction de la contrainte principale en cours), OBJET est créé dans le modèle, et le système traite une nouvelle construction du même type,

- si l'utilisateur montre le menu SUIVANT, le système choisit une autre solution plausible pour OBJET et la propose (l'affiche). Il se replace en situation de PROPOSITION DE SOLUTION,

- si l'utilisateur montre le menu ABANDON, le système abandonne la solution (rien n'est créé dans le modèle) et attend une nouvelle construction du type de celle qui était en cours,

- si l'utilisateur montre un menu DIFFERE, l'OBJET est adopté et créé dans le modèle.

Considérons deux cas de construction sous contraintes: création d'un segment tangent à deux cercles et création d'un segment passant par deux points.

a) segment tangent à deux cercles :

L'opérateur a précisé qu'il veut créer un segment, tangent à deux cercles, et il a désigné deux cercles. La primitive d'INTERACTION, outre le nom (dans le modèle) des objets cercles montrés, retourne les coordonnées du point qui a servi à l'identification (Cf figure 1).

A partir de ces éléments, le système peut récupérer les paramètres des deux cercles (centres et rayons) dans le modèle et calculer les tangentes possibles(figure 2). A partir des deux points ayant servi à l'identification, il choisira la solution indiquée sur la figure 3, qui semble la plus plausible. Si l'utilisateur n'est pas satisfait de cette solution, il pourra demander l'une après l'autre l'affichage de toutes les solutions, jusqu'à satisfaction ou abandon (on boucle sur les solutions).

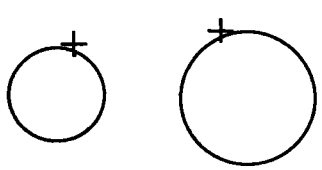


Figure 1

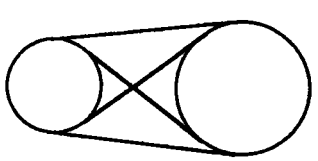


Figure 2

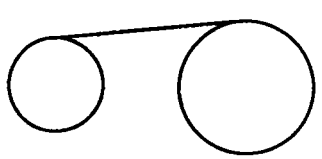


Figure 3

b) segment passant par deux points :

Il est très facile d'autoriser la définition des extrémités, non seulement par des coordonnées, mais également par n'importe quel type de construction sous contrainte. Il suffit de déclarer les menus correspondants à des constructions de points sous contraintes comme locaux. A chaque appel à la primitive INTERACTION, l'utilisateur peut sélectionner la contrainte de son choix (par exemple MILIEU d'un segment existant) et obtenir le point cherché.

c) ZOOM :

Il est possible à tout moment dans une construction d'agrandir une partie de la scène, sans abandonner la construction en cours. Il suffit de déclarer le menu ZOOM comme immédiat pour toutes les constructions sous contraintes.

I.2. Dialogue pour les contours.

I.2.1. Les contours évidents.

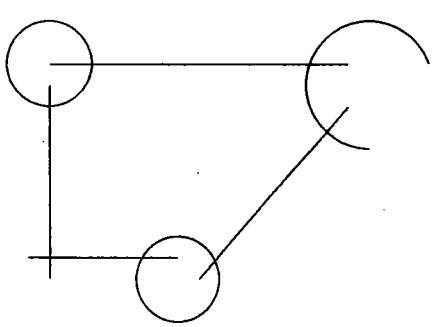
Cette forme de construction permet dans certaines conditions de réaliser un contour en un minimum d'interactions, soit, dans le cas présent en désignant un segment du contour et le sens de parcours de celui-ci. Il faut, pour cela, que la suite d'objets sur laquelle s'appuie le contour, réponde aux deux seuls critères suivants :

- chaque objet ne doit être coupé que par deux autres objets visibles.

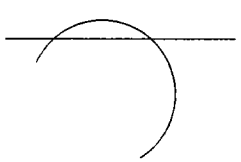
- il n'y a pas d'intersections multiples.

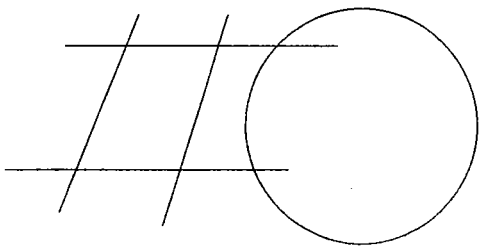
Nous avons jugé cette option fort utile dans la mesure où la personne établirait sa scène dans le but précis de construire son contour, celle-ci se rapprochant donc du contour voulu.

scène type :



situations interdites :





Reste le problème lié à l'orientation des cercles : ceux-ci sont parcourus dans le sens trigonométrique, donc selon le sens de parcours choisi au départ on obtiendra soit la figure 1, soit la figure 2.

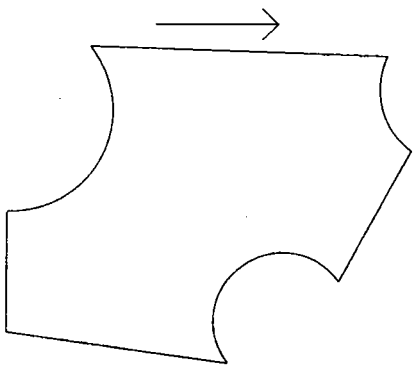


figure 1

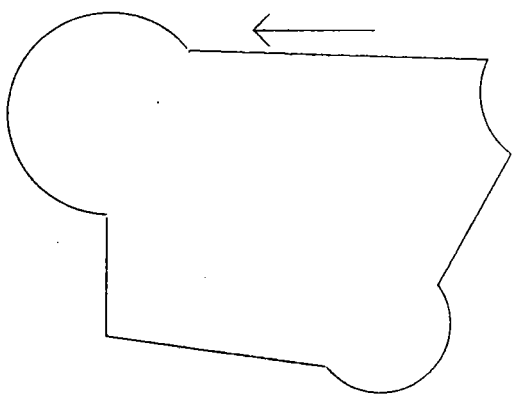


figure 2

Un menu permet, le cas échéant, d'inverser le sens de parcours d'un ou plusieurs cercles par simple désignation.

I.2.2. Les contours généraux.

Nous avons cherché, de même que précédemment, le moyen le plus simple et le plus naturel de désigner un ensemble d'objets afin de construire n'importe quel contour. Cette désignation est faite à l'aide de la primitive "INTERACTION".

Les difficultés sont liées au problème de spécification du sens de parcours des éléments et du choix de l'intersection d'un élément avec son suivant en cas d'ambiguïté. Une stratégie simple permet de réaliser n'importe quel contour en ne désignant, dans la plupart des cas, qu'une seule fois un objet, et au pire deux fois.

L'utilisation de la primitive "INTERACTION" pour la désignation et la spécification de certains menus comme "IMMEDIAT" de CONTOUR, fournissent à l'utilisateur des outils puissants et pratiques.

On peut donc, lors de la désignation des objets du contour, avoir accès à différentes fonctions sans quitter l'opération d'entrée des données.

Exemples :

- le zoom Lors de la saisie, certaines parties du dessin peuvent être confuses et ne pas permettre une désignation précise. L'utilisateur peut alors abandonner momentanément l'opération en cours, agrandir la portion désirée, poursuivre sa saisie, puis revenir à l'espace de base afin de finir son opération de désignation.

- les constructions sous contraintes (I.1.) En cours de saisie, l'utilisateur peut, à tout moment, interrompre celle-ci afin de rajouter un éventuel élément. Cette construction sera, en tout point, identique à celle faite dans d'autres conditions. L'objet ainsi construit pourra alors être intégré au contour au même titre que ceux déjà existant. L'opération terminée,la saisie reprendra auniveau du dernier objet désignépar l'utilisateur.

I.3 Dialogue de la cotation

Dans la maquette 2D de SACADO, les cotations portent sur un ou deux objets simples : points, segments, cercles et arcs.

Elles ont fait l'objet d'un effort particulier en matière d'interactivité qui s'est traduit par :

- l'absence délibérée de menus (voir ci-dessous les conséquences de l'absence de menus),

- un affichage dynamique commandé par une souris (à rapprocher de l'affichage élastique d'un segment),

- le défilement des différentes solutions possibles si la première proposée, qui paraissait la plus vraisemblable, n'est pas celle souhaitée par l'utilisateur,

- le déplacement, avec le même affichage dynamique, d'une cotation existante,

- la possibilité de forcer une cote à une valeur différente de celle calculée,

- la possibilité d'afficher une cotation provisoire (demande de renseignements).

Les cotations devaient également permettre de tester et de valider certaines facilités de dialogue du logiciel, à savoir :

a - pouvoir abandonner l'action à tout moment

b - pouvoir suspendre (et non abandonner) une quelconque action le temps d'en réaliser une autre.

EXEMPLE : cotation entre deux cercles.

L'utilisateur

(1) montre le menu 'COTATIONS'

(2) montre un cercle existant
OU
en crée un sans pour autant abandonner l'action en cours et devoir recommencer en (1) (rendu possible par (b))

(3) montre un deuxième cercle
OU
en crée un sans pour autant abandonner l'action en cours et devoir recommencer en (1) (rendu possible par (b))

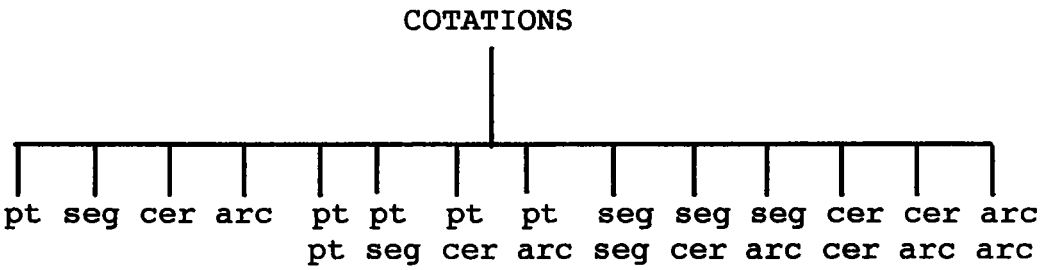
(4) voit s'afficher une cotation indiquant la distance entre les centres (entraxe).

(5) change, en déplaçant la souris, la position de la valeur de cote et, avec elle, l'ensemble de la cotation jusqu'à ce qu'il la juge bien située (pas de chevauchement avec la scène...) ET, le cas échéant, appuie sur la barre espace pour voir défiler les autres cotations possibles. à savoir, dans notre exemple, la distance entre les points les plus éloignés des deux cercles, celle entre les deux points les plus proches, etc...
ET / OU
tape une série de caractères qui s'ajoutent aux pré et post textes.

(6) valide son choix avec un bouton de la souris.

CONSEQUENCES DE L'ABSENCE DE MENUS

Nous avons choisi de ne pas encombrer les cotations d'un menu arborescent du type :



Nous avons en effet préféré, parce que cela nous semble d'un usage plus pratique, que le logiciel déduise automatiquement du nombre et de la nature du (des) argument(s) désigné(s), le type de cotation souhaité.

Nous avons en effet préféré, parce que cela nous semble d'un usage plus pratique, que le logiciel déduise automatiquement du nombre et de la nature du (des) argument(s) désigné(s), le type de cotation souhaité.

Privé du contexte d'un sous-menu, on ne connait donc pas à l'avance le nombre d'objets à coter ni, par conséquent, le nombre d'appels nécessaires à la primitive "INTERACTION".

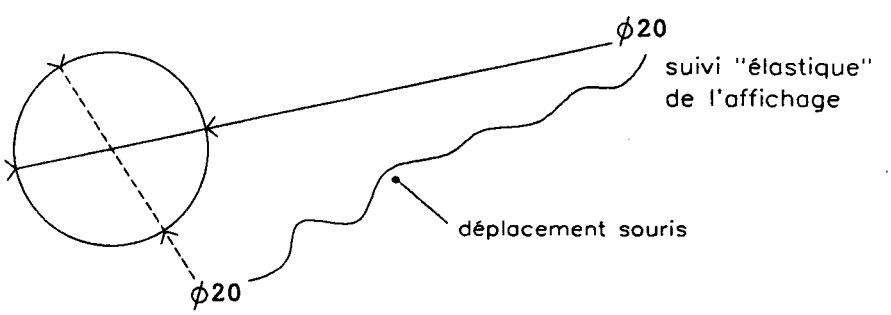
Ce problème a automatiquement trouvé sa solution dans le fait que toutes les interventions de l'utilisateur se font par l'intermédiaire d'UNE SEULE primitive retournant autant d'informations que possible (objets récupérés, coordonnées du pointé,...) : à la deuxième interaction, il suffit donc de tester les paramètres retournés : si aucun objet n'est récupéré, l'utilisateur a sous-entendu, en montrant la position initiale de la cote, que la cotation ne mettra qu'un élément en jeu; si par contre on recueille un objet, on en prend note et c'est le pointé qui a servi à désigner l'objet qui donne aussi la position initiale de la cote.

Il faut remarquer que ce principe, qui évite plusieurs validations à l'utilisateur sans rendre sa tâche plus contraignante, peut se généraliser à un nombre quelconque d'arguments.

AFFICHAGE DYNAMIQUE.

Par affichage dynamique, nous entendons que l'utilisateur, après qu'il ait indiqué l'endroit où doit s'afficher la valeur de cote, peut encore, notament pour des raisons d'esthétique, déplacer l'ensemble du tracé (traits de rappel, valeur, pré et post-textes) en bougeant la souris, cette dernière restant liée au point d'affichage de la valeur de cote.

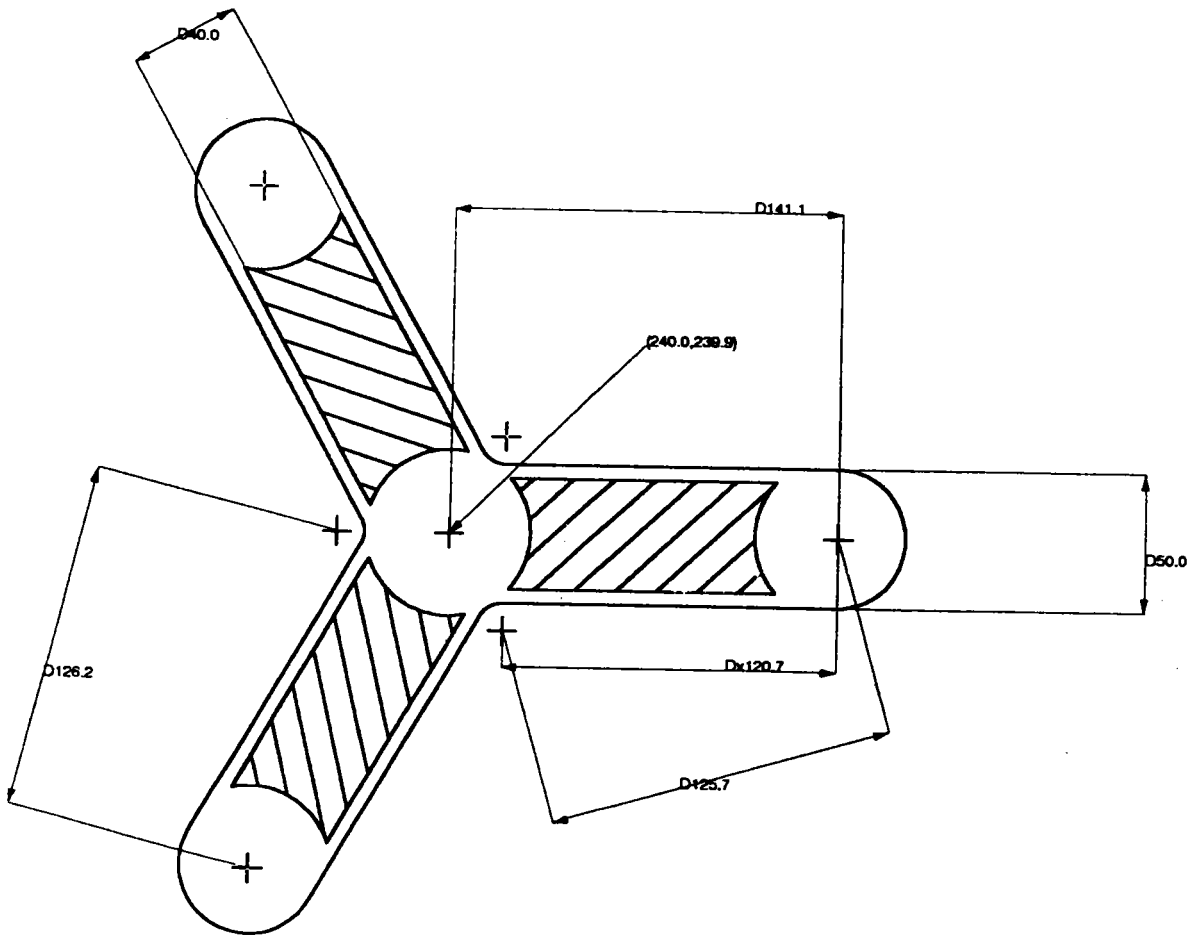
Exemple :



position initiale de la souris

Il est donc possible de situer exactement, rapidement et d'une manière très agréable les tracés de cotation.

Exemple de plan réalisé avec SACADO 2D



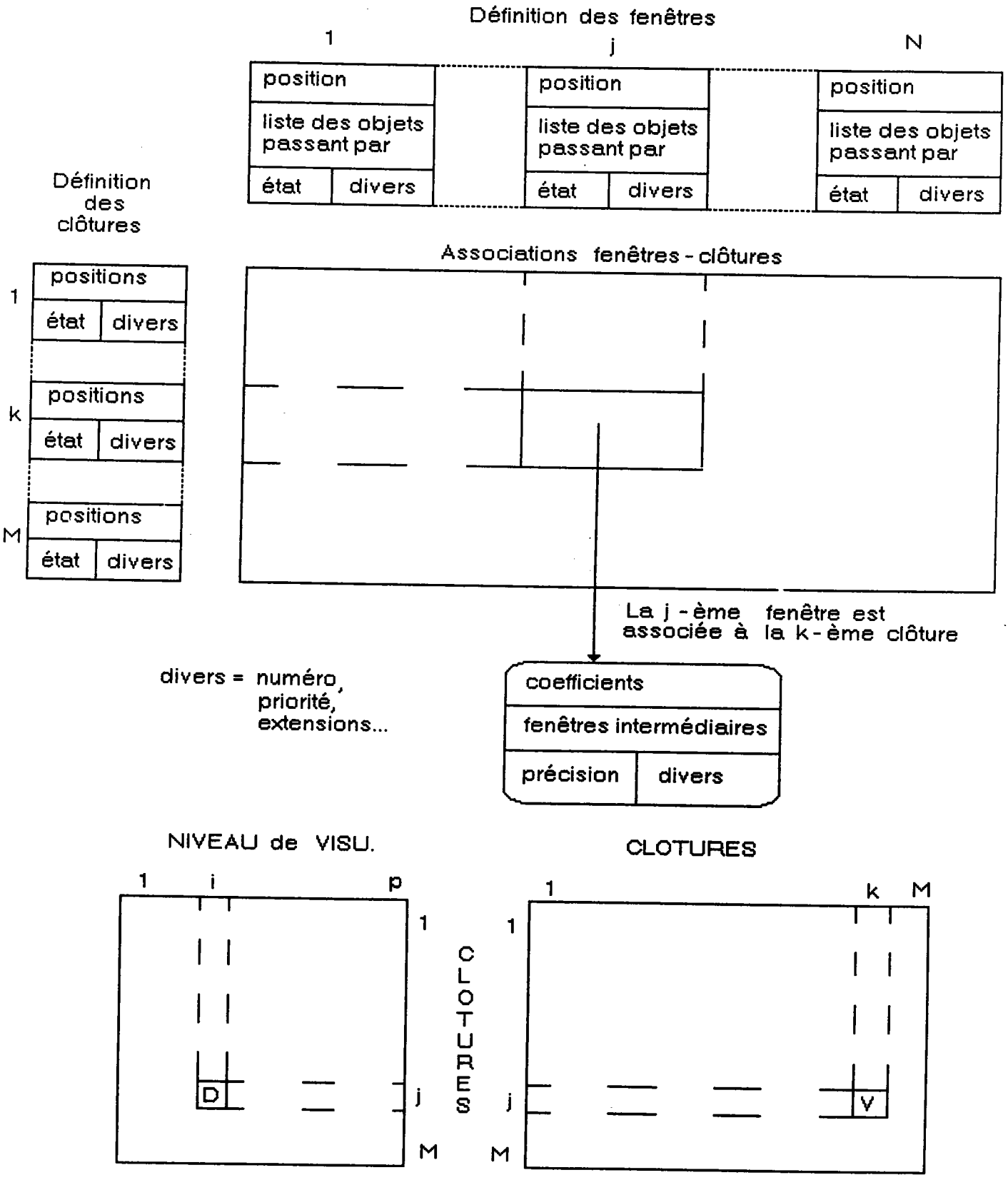
ANNEXE 2

Dans cette annexe, nous donnons des schémas simplifiés de structures nécessaires à la gestion du multi-fenêtrage (affichages, position du curseur, passages fenêtres<-->clôtures, ...). Nous présenterons également des copies d'écran de l'environnement "multi-fenêtré" tel qu'il est proposé à l'opérateur.

Les figures ci-après montrent :

- une table des fenêtres,
- une table des clôtures,
- une matrice des associations fenêtres-clôtures,
- une matrice des associations niveaux de visualisation-clôtures (Détectable, Visible, Invisible),
- une matrice des interférences entre clôtures (Vrai ou Faux).

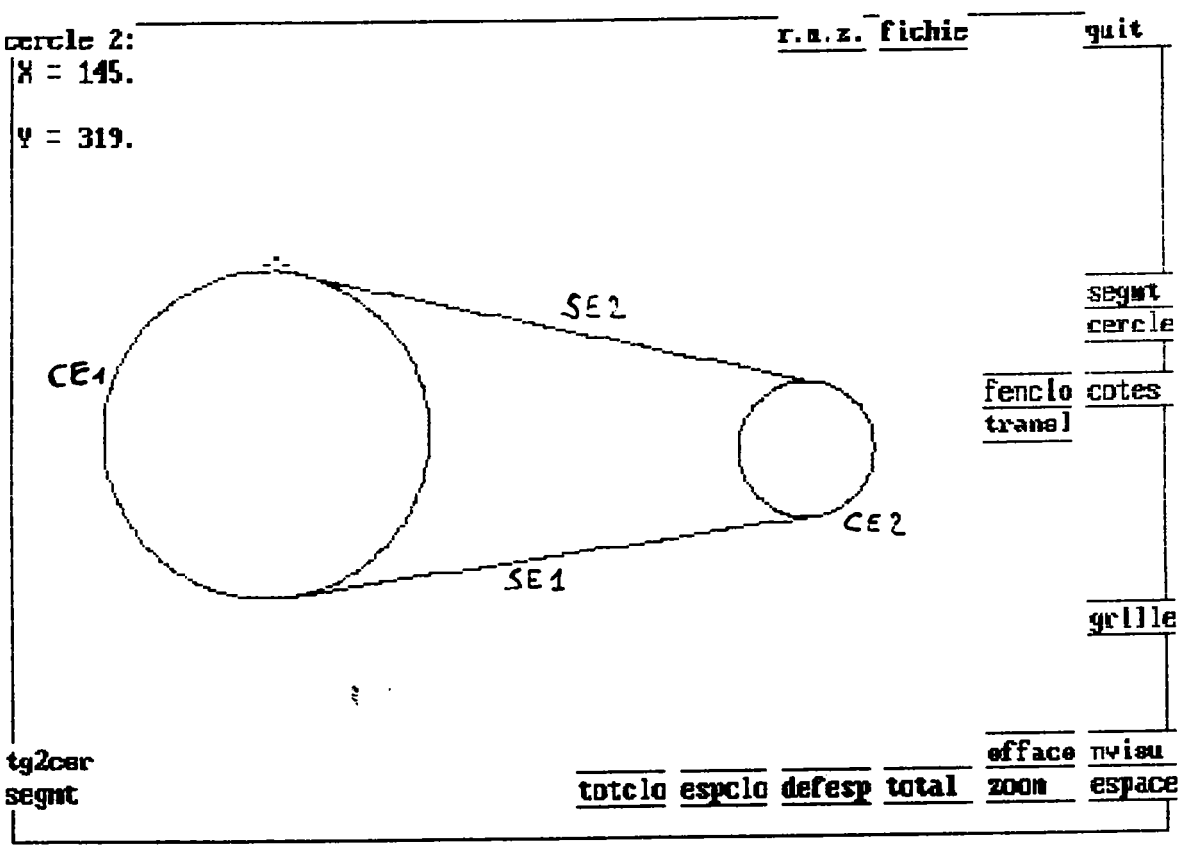
Il faut noter que ces schémas sont présentés à titre indicatif, afin de montrer la quantité et la nature des informations qui sont conservées dans le modèle de visualisation. Notamment, nous avons utilisé le terme "divers" pour indiquer que d'autres attributs, que ceux qui apparaissent explicitement, sont (ou peuvent être) utiles à la définition des différentes entités.



Le niveau de visualisation i est Désignable dans la clôture j qui interfère avec clôture k.

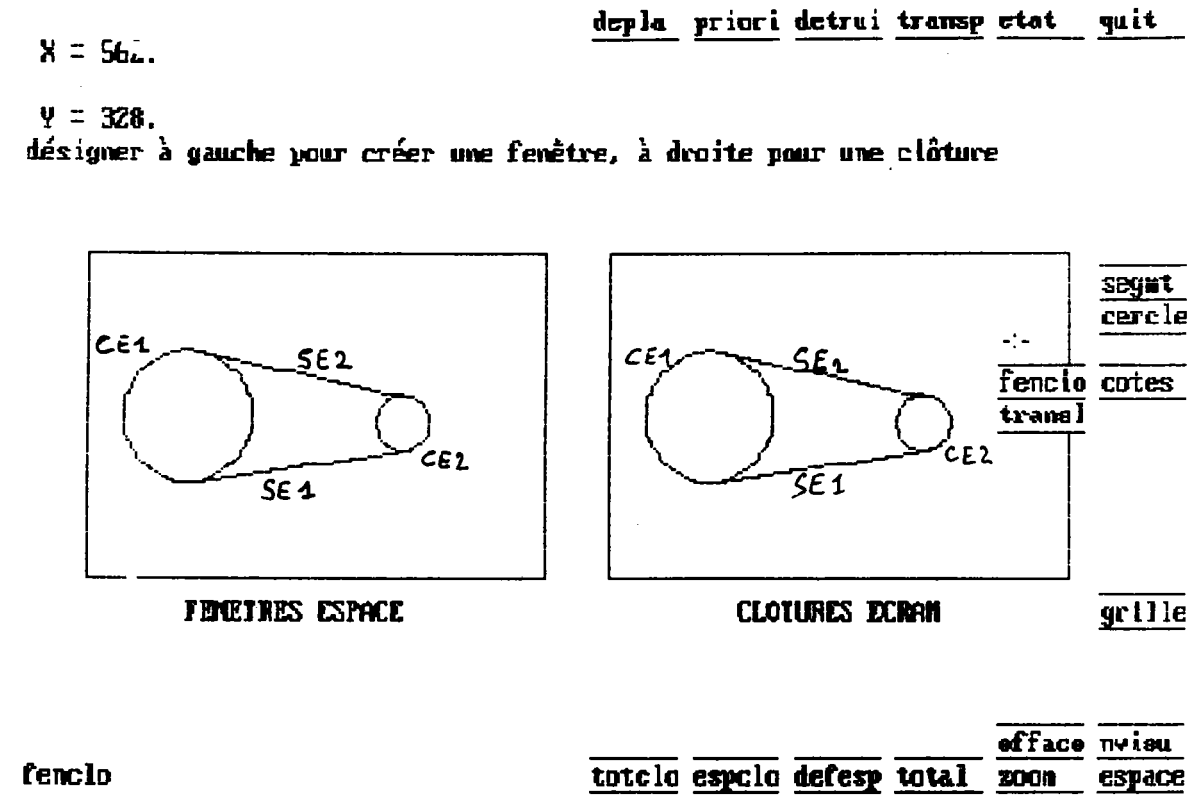
Les copies d'écran qui suivent donnent un aperçu des possibilités offertes par l'environnement "multi-fenêtré". Une première partie est consacrée aux options qui permettent la spécification interactive d'un tel environnement. Nous terminons par des exemples qui peuvent justifier l'utilisation de plusieurs clôtures (et plusieurs fenêtres).

La figure A2.1. montre une scène (simple) représentée sur tout l'écran.



(figure A2.1.)

L'option "fenclo" permet de définir et de manipuler des fenêtres et des clôtures. La sélection de cette option conduit à la situation de la figure A2.2. L'écran présenté à l'utilisateur est constitué de deux clôtures (propres au système). La clôture de gauche représente l'espace utilisateur, celle de droite visualise l'écran.



(figure A2.2.)

Ainsi, l'utilisateur peut voir comment le monde réel est représenté à l'écran. Ici, tout l'espace est représenté sur tout l'écran. A partir de là, on peut construire des cadres dans l'une ou l'autre des deux clôtures : à gauche pour définir des fenêtres, à droite pour définir des clôtures.

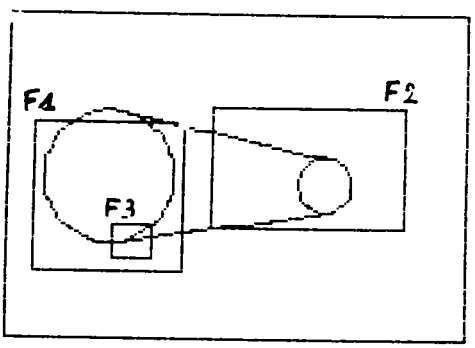
Supposons que l'on définisse trois fenêtres F1, F2, et F3, et quatre clôtures C1, C2, C3 et C4. On associe (par exemple) F1 à C2, F2 à C3 et F3 à C1 (C4 n'est associée à aucune fenêtre). La correspondance espace-écran est immédiatement visualisée (cf. figure A2.3.).

depla priori detrui transp etat quit

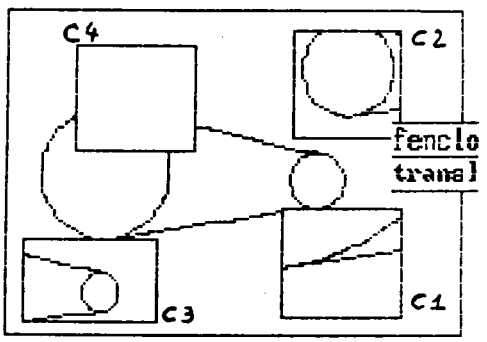
X = 266.

Y = 399.

creer une clôture



FENETRES ESPACE



CLOTURES ECRAN

segmt
cercle
fenclo cotes
transl
grille

fenclo

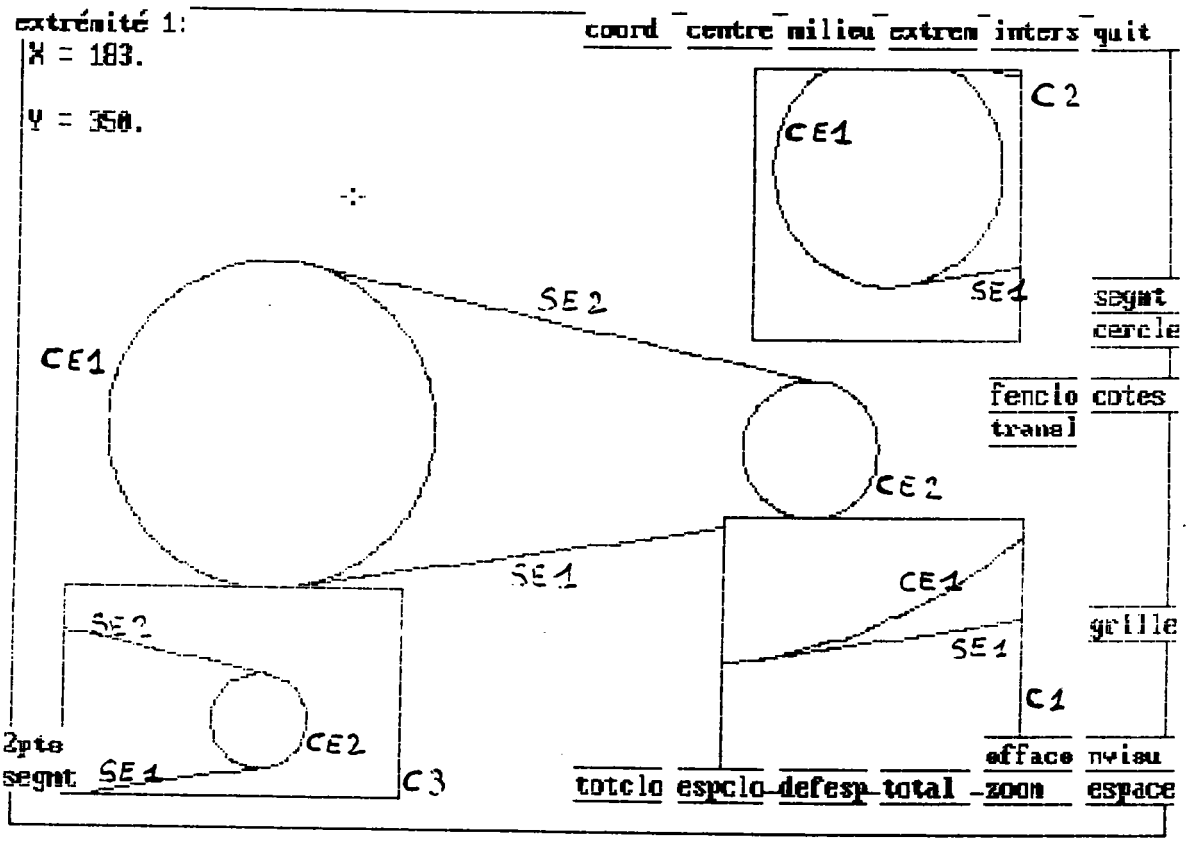
efface nveau
totclo espclo defesp total zoom espace

(figur A2.3.)

L'option "fenclo" possède des Locaux (en haut à droite de l'écran):

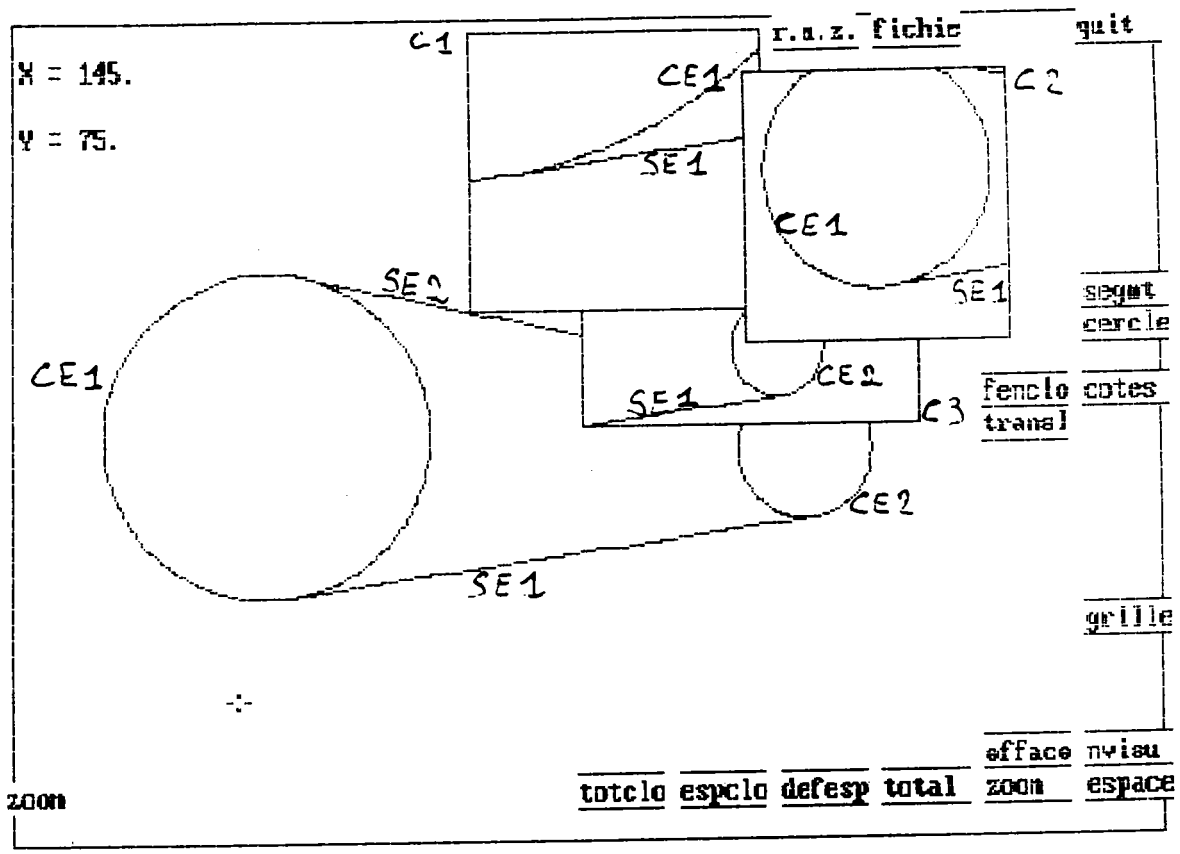
- "depla", pour déplacer des fenêtre ou des clôtures,
- "priori", pour changer les priorités relatives des clôtures,
- "détrui", pour détruire des fenêtres ou des clôtures (et les associations qui s'y rattachent),
- "transp", pour rendre transparente une clôture dans le cas où elle cache des parties intéressantes de l'écran,
- "etat", pour associer des fenêtres et des clôtures, ou fixer l'état d'une clôture (Détectable, Visible ou Invisible).

Si l'on suppose que les clôtures C1, C2 et C3 soient rendues "Détectables", et que C4 soit "Invisible", lorsque l'on continuera à travailler sur la scène (par sélection d'un autre menu que "fenclo" ou que ses Locaux"), l'écran proposé à l'utilisateur sera celui de la figure A2.4.



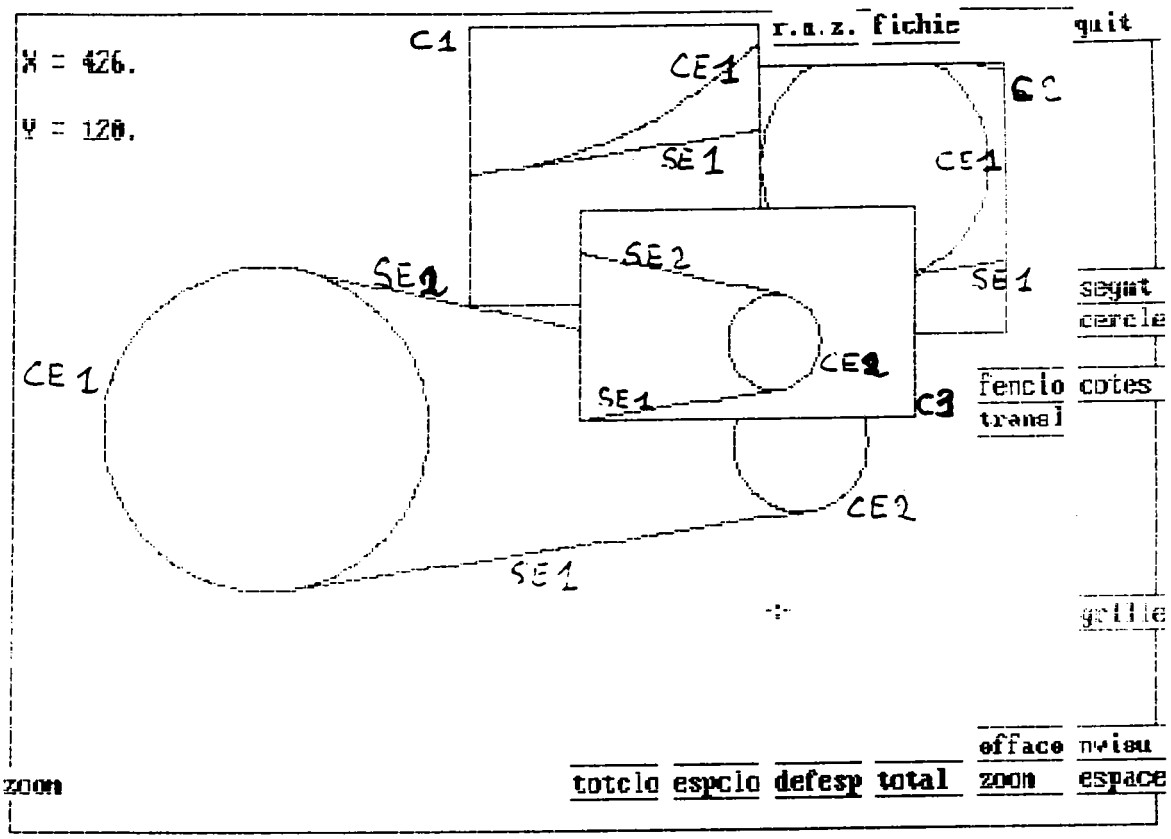
(figure A2.4.)

A tout moment, on peut changer cette situation en revenant à l'option "fenclo". La figure A2.5. montre l'écran après des déplacements de clôtures.



(figure A2.5.)

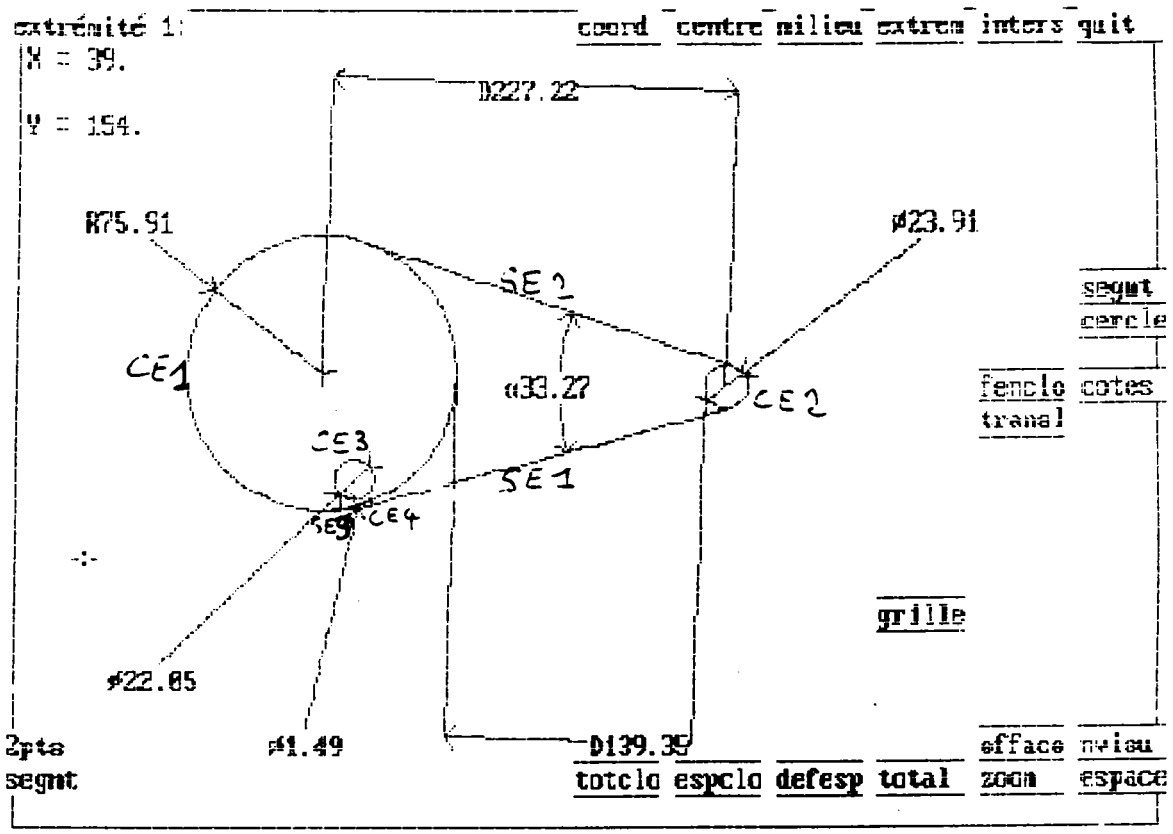
De façon analogue, il est facile de modifier les priorités relatives des clôtures. La figure A2.6. montre ce que pourrait donner une telle manipulation, à partir de la situation de la figure A2.5.



(figure A2.6.)

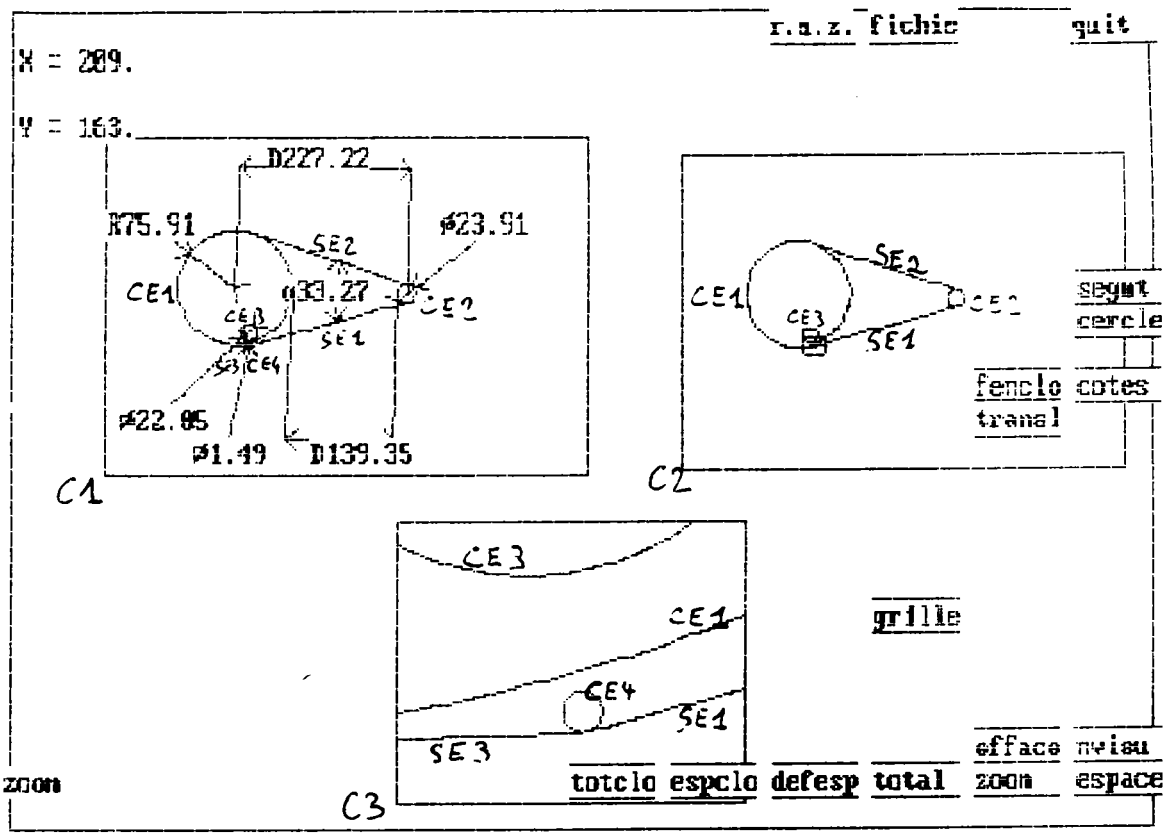
Nous ne détaillerons pas toutes les possibilités qu'offre l'option "fenclo". Notamment, cette option permet des associations multiples (superpositions de parties différentes de l'espace dans une seule clôture), ainsi que des relations entre les niveaux de visualisation et les clôtures. Il nous semble préférable de donner des exemples de l'intérêt du modèle de visualisation tel que nous l'avons défini.

Une scène peut vite devenir difficilement utilisable, parce que trop chargée ou parce que certains détails sont trop petits (et donc se voient mal) dans l'ensemble. La figure A2.7. donne un exemple simpliste d'une telle situation.



(figure A2.7.)

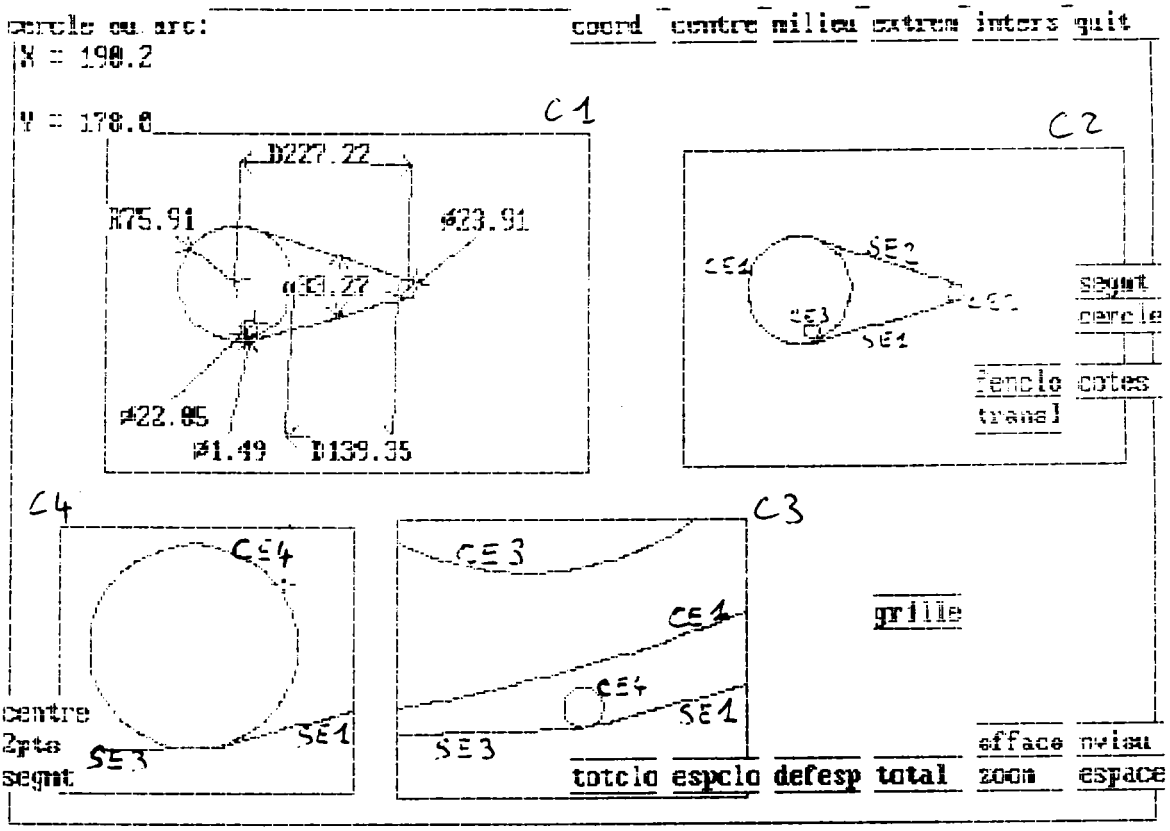
Dans un cas comme celui-ci, on aimerait pouvoir visualiser toute la scène telle quelle (pour avoir sous les yeux un maximum d'informations), en même temps que la scène sans habillage (pour continuer à travailler avec l'essentiel), et avoir un agrandissement d'une zone confuse (pour mieux la voir et pour pouvoir travailler dessus). Grâce à loption "fenclo", on peut (par exemple) définir trois clôtures et une fenêtre, de manière à obtenir l'écran de la figure A2.8.



(figure A2.8.)

Comme le montre cette figure, on a décidé d'associer tout l'espace réel aux clôtures C1 et C2, et la fenêtre (région traversée par les cercles CE1, CE3, CE4 et par les segments SE1 et SE3) à la clôture C3. De plus, l'habillage est dans un niveau de visualisation différent de celui des objets géométriques. Ce niveau est associé à la clôture C1, mais pas aux clôtures C2 et C3.

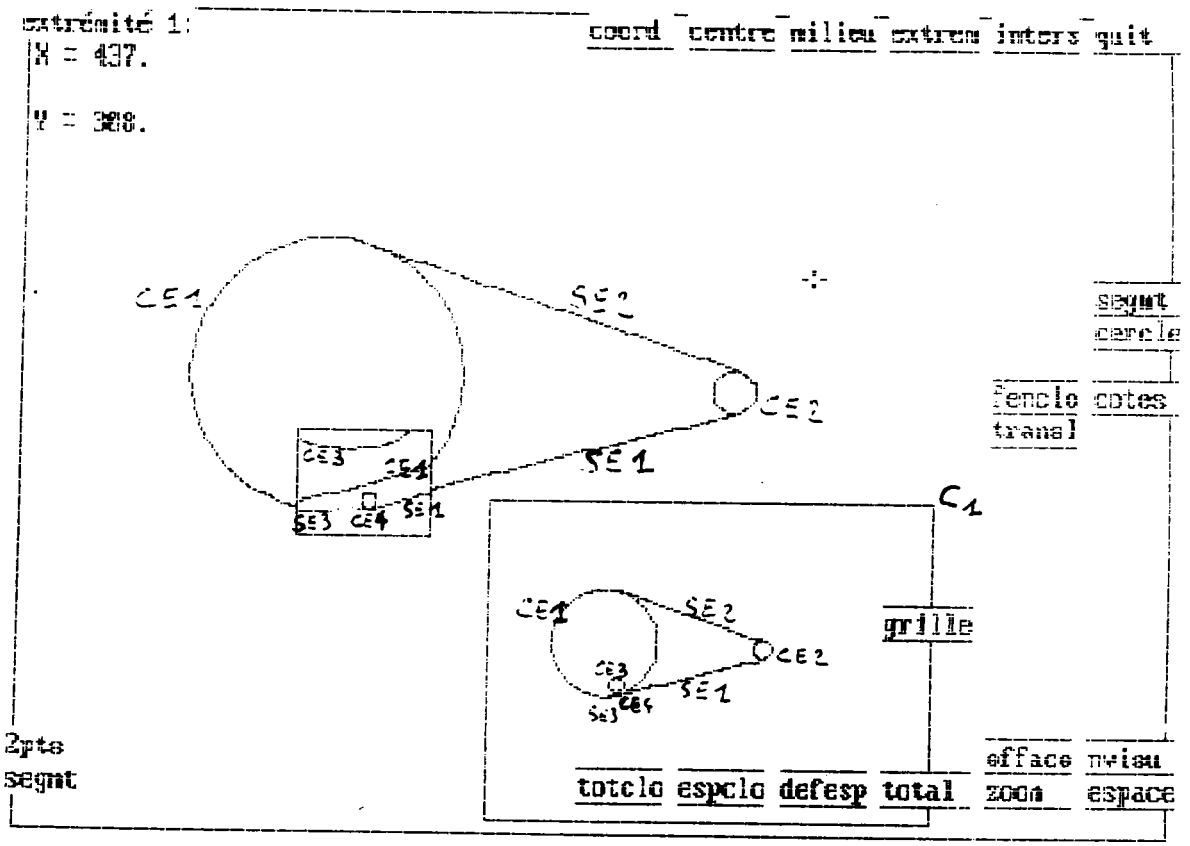
On peut associer une fenêtre et une clôture supplémentaires pour avoir un plus fort grossissement de la région du cercle CE4 (cf. figure A2.9.).



(figure A2.9.)

Notons qu'à présent il est facile de construire (par exemple) un segment dont les extrémités sont les centres des cercles CE2 et CE4, puisque l'on peut désigner CE2 dans la clôture C2, et CE4 dans l'une des clôtures C4 ou C3 (au choix).

La figure A2.10. donne un dernier exemple des possibilité offertes à l'utilisateur. La totalité d'une scène est visualisée sur tout l'écran et sur une clôture C1. La zone confuse est placée à la position même de la région agrandie. On a, ainsi, l'illusion d'un effet loupe.



(figure A2.10.)

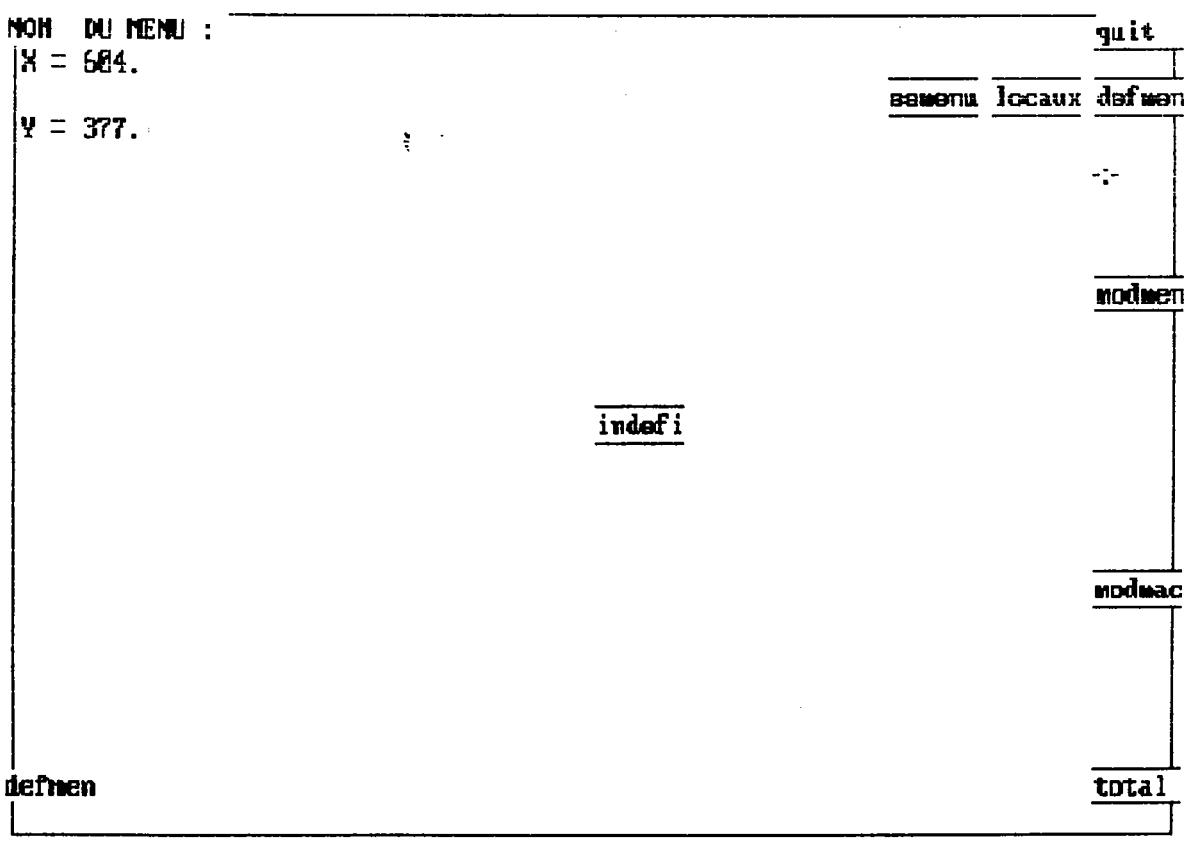
On peut imaginer une infinité de situations. Nous terminerons en rappelant, qu'en dehors de sa contribution à la convivialité de l'affichage et du dialogue, le modèle de visualisation permet de réduire les temps de réponse, si la scène est intelligemment découpée.

ANNEXE 3

Dans la quatrième partie, nous avons présenté la maquette de l'application : "concepteur d'interface". Dans cette annexe, nous donnons un aperçu de son utilisation. Cette application est constituée de trois options principales :

- "defmen", pour la définition rapide d'une option menu (nom du menu, position à l'écran et action associée),
- "modmen", pour affiner la définition d'une option menu,
- "modmac", pour la définition d'un macro-menu (nom du macro-menu, position à l'écran et séquence des composants).

La figure A3.1. montre l'écran proposé à la suite de la sélection de l'option "defmen". Celle-ci invite l'opérateur à nommer une nouvelle option menu ("indéfini", par défaut). Puis, l'opérateur aura la possibilité de fixer la position écran du nouveau menu, et, enfin, il pourra spécifier le nom (externe) de l'action associée. Il faut noter qu'il est également possible de définir des niveaux (arborescences ou locaux) grâce aux menus Locaux de "defmen", à savoir : "ssmenu" et "locaux". Si l'opérateur saute une étape de la spécification, c'est la définition par défaut qui est retenue (nom : "indéfini", position : centre de l'écran, action : "indéfinie").

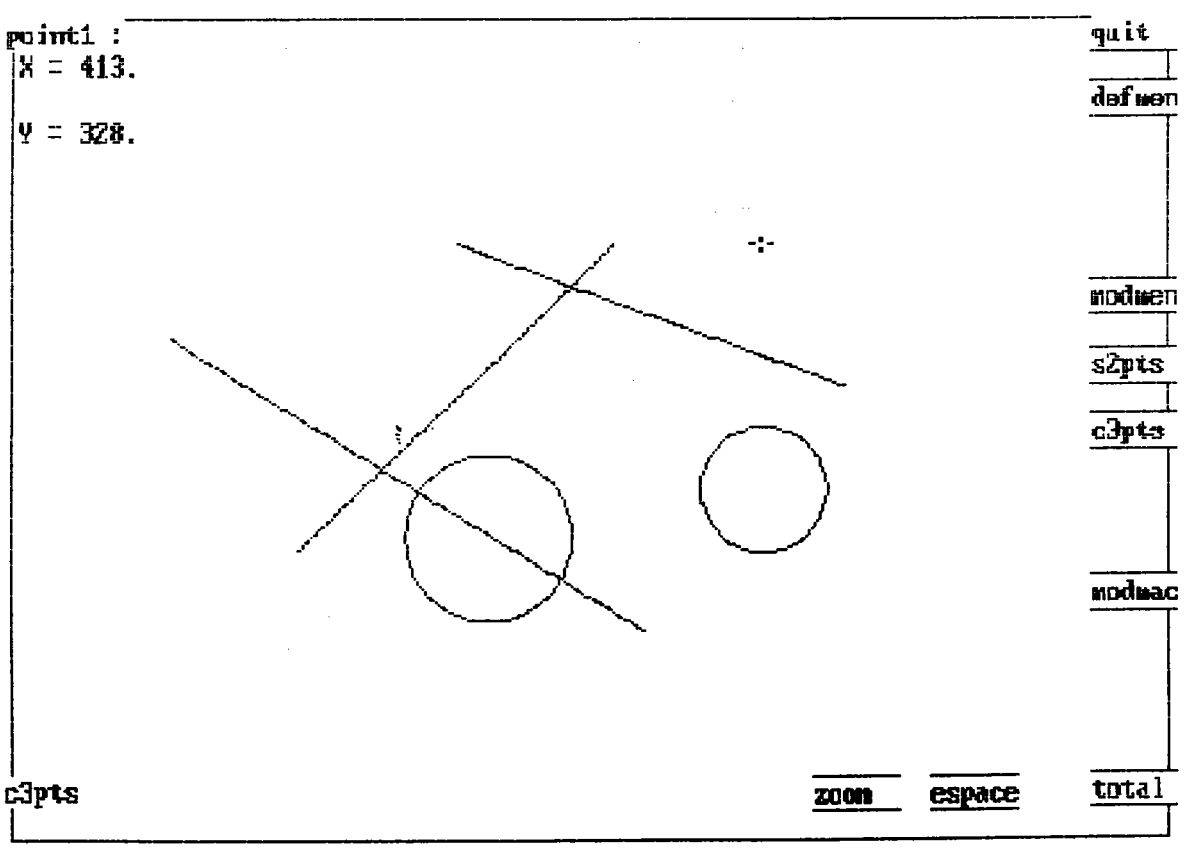


(figure A3.1.)

Supposons que le concepteur ait défini les quatre options suivantes :

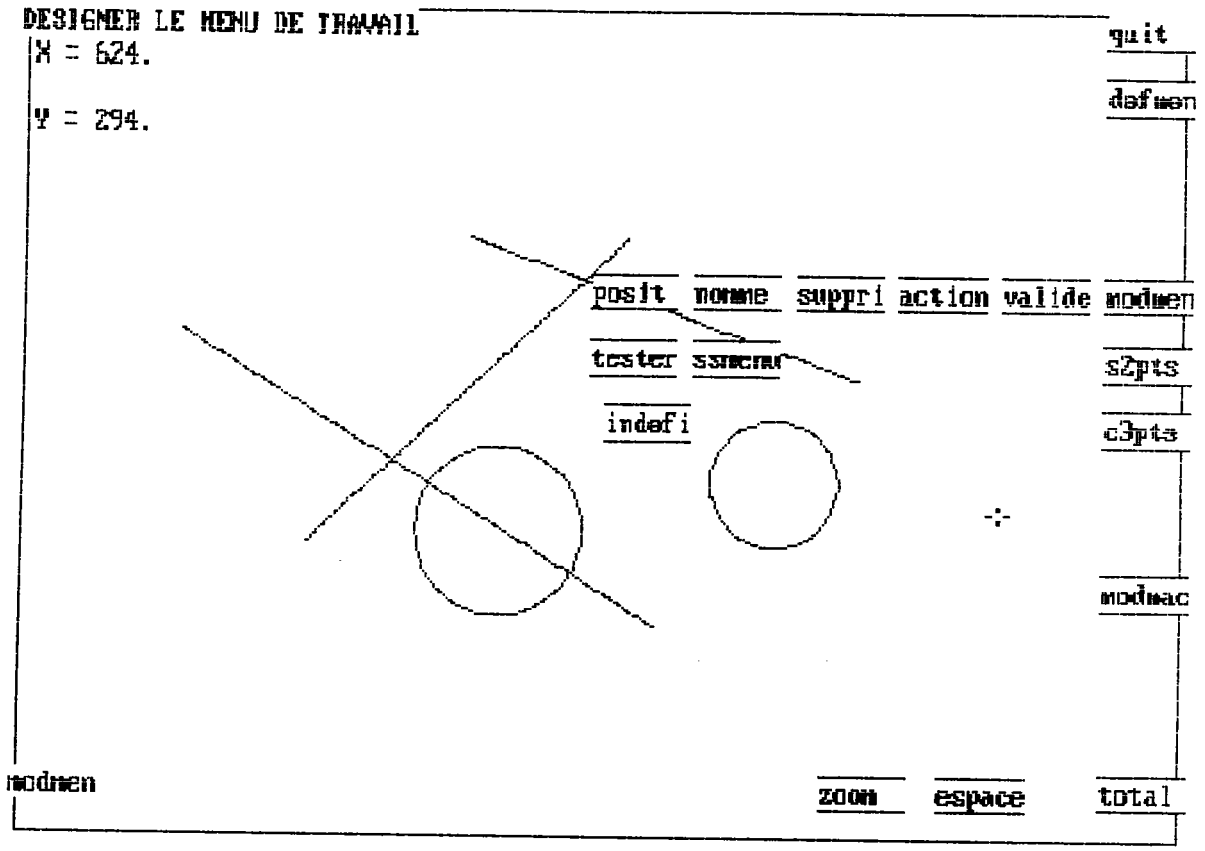
- "s2pts", pour la construction d'un segment par ses extrémités,
- "c3pts", pour la construction d'un cercle passant par trois points,
- "zoom", pour pouvoir effectuer un zoom,
- "espace", pour revenir sur un zoom.

Dans la mesure où la primitive "APPEL" connait les noms internes et externes des action associées à ces menus, et qu'un programmeur a défini le code correspondant, on peut commencer à construire une scène (par exemple, celle de la figure A3.2.).



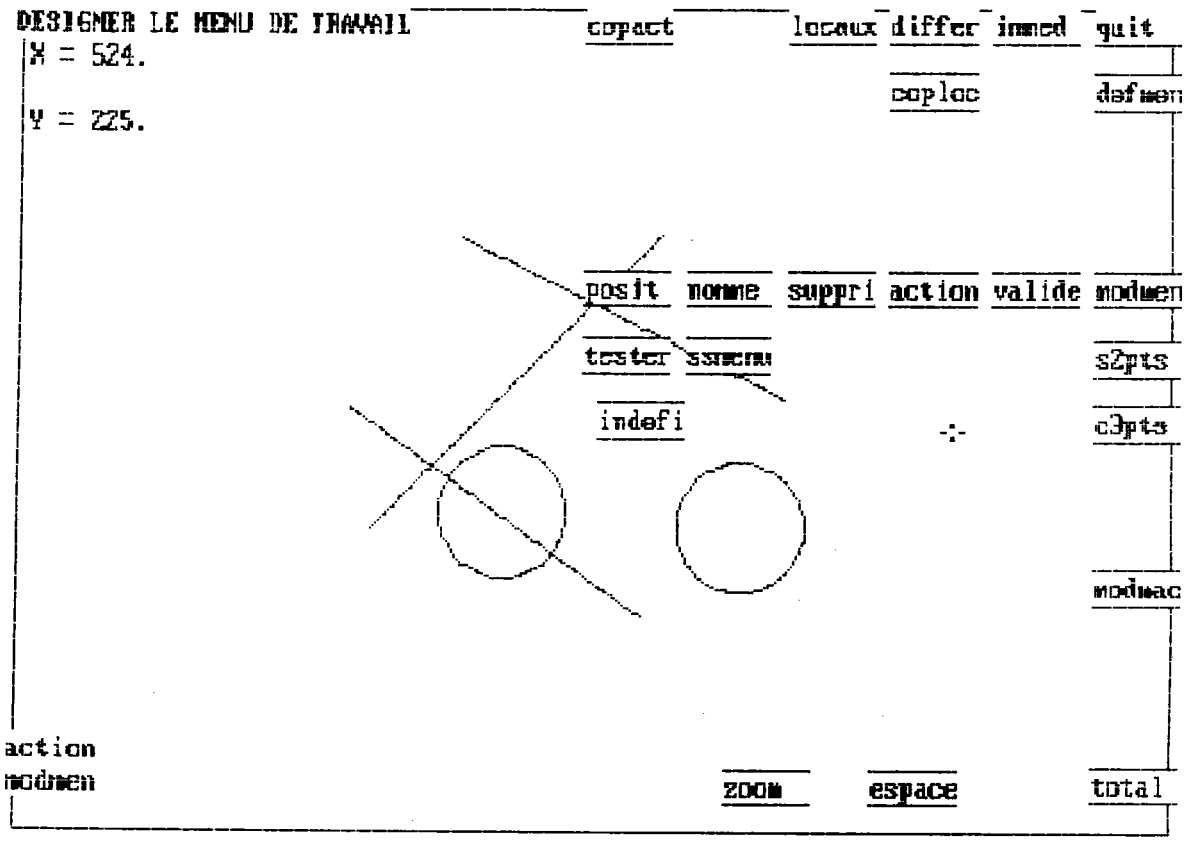
(figure A3.2.)

Si l'on désire affiner la définition des menus, il suffit de sélectionner l'option "modmen". Comme le montre la figure A3.3., cette option possède des Locaux qui permettent de manipuler les différents attributs d'une option menu (position, nom, action, ...).



(figure A3.3.)

Le menu "action" (Local de "modmen") possède des Locaux qui permettent, entre autres, de spécifier des liens de compatibilité et de localité (cf. figure A3.4.).

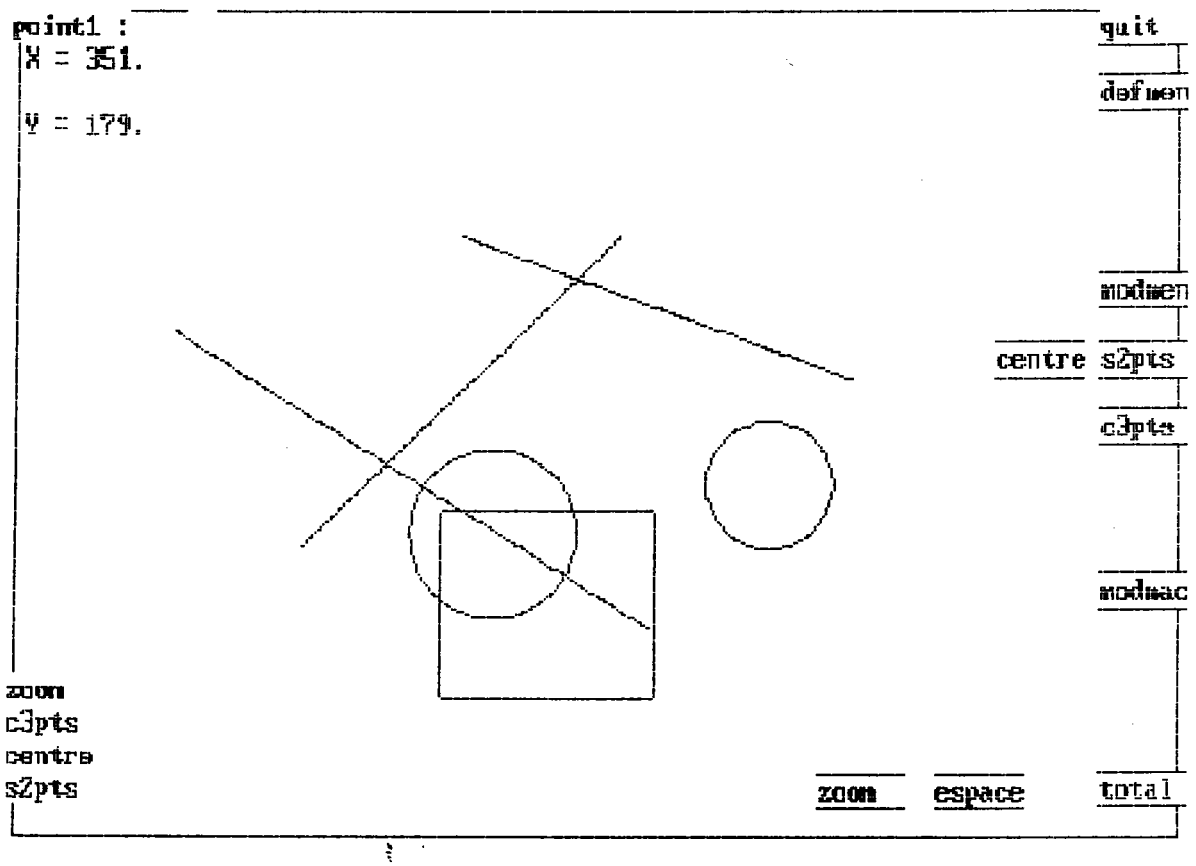


(figure A3.4.)

Supposons que l'on ait spécifié :

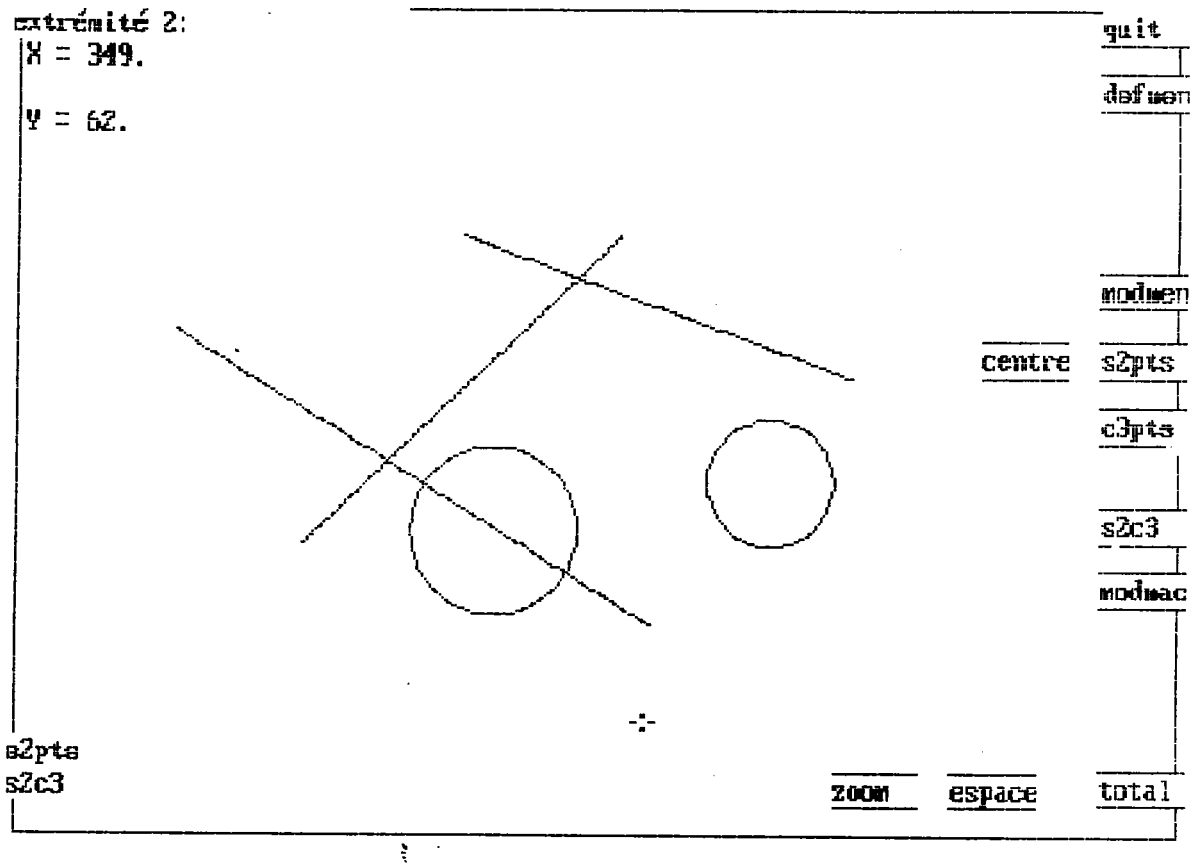
- "centre" (construction d'un point centre d'un cercle), Local de "s2pts",
- "c3pts", Immédiat de "centre",
- "zoom", Immédiat de "c3pts".

La figure A3.5. montre (en bas à gauche de l'écran) qu'il est alors aisé d'effectuer un zoom lors de la création d'un cercle (passant par trois points) dont le centre deviendra l'une des extrémités d'un segment.

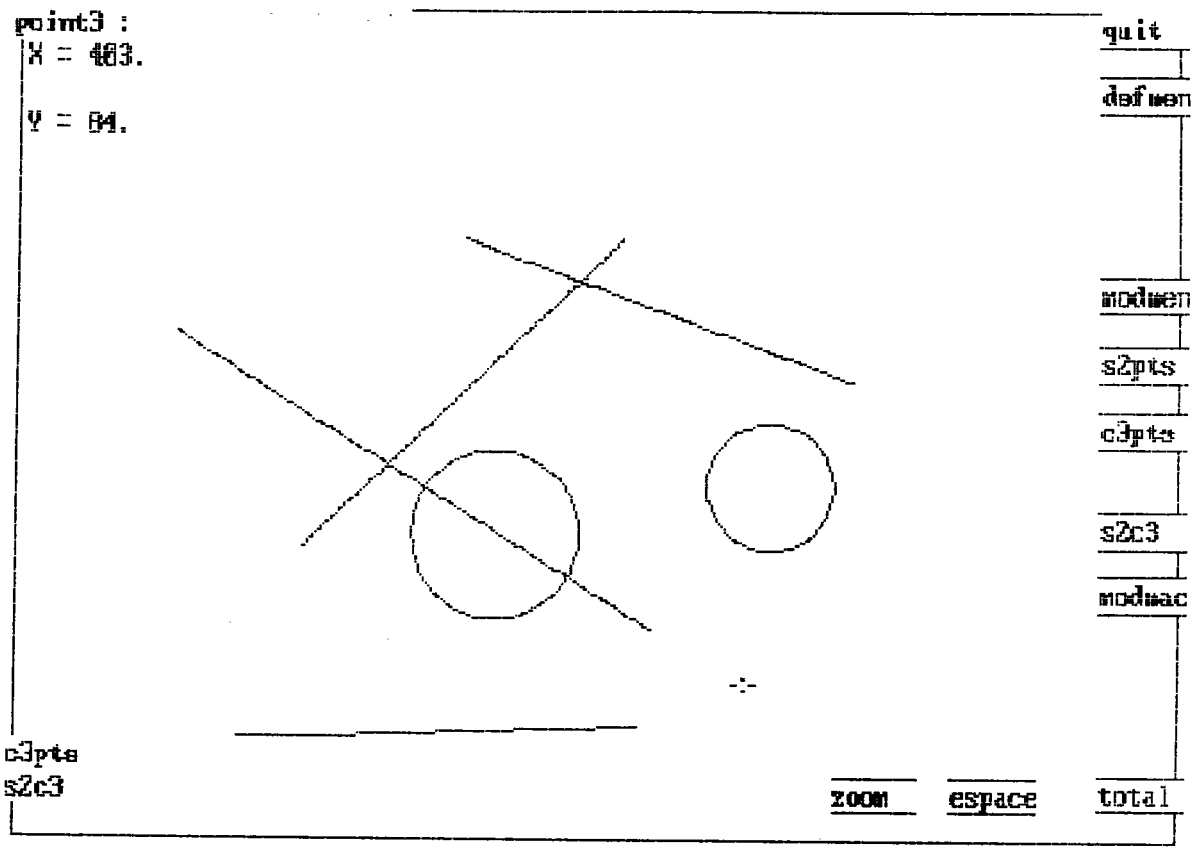


(figure A3.5.)

L'utilisation de la dernière option, "modmac", permet (par exemple) de définir un macro-menu, "s2c3", comme étant la séquence des options "s2pts" et "c3pts". A la suite de quoi, l'unique sélection de "s2c3" lancera automatiquement la séquence. La figure A3.6. montre le premier composant ("s2pts") en cours d'exécution. De même, la figure A3.7. montre l'exécution du second composant ("c3pts").


extrémité 2:
X = 349.
Y = 62.
quit
defmen
modmen
centre
s2pts
c3pts
s2c3
modmac
s2pts
s2c3
zoom
espace
total

(figure A3.6.)



(figure A3.7.)

Cette annexe visait à montrer ce que pourrait être une application destinée au concepteur. Elle a donné un aperçu des facilités de prototypage. Il est important de rappeler que cette application est définie dans l'environnement SACADO. Ce point est essentiel compte tenu de notre objectif d'intégration.

BIBLIOGRAPHIE

[ARM 88] : Analyse Recherche et Méthode Conseil
" Cahier des Charges de SACADO "
(Document interne, Septembre 86)

[ANS 82] : E. ANSON
" The Device Model of Interaction "
(Computer Graphics, 16-3, Juillet 82, p. 107-114)

[ANV 89] : RAPPORT ANVAR 1989
" SACADO : Système de C.F.A.O. ouvert sur micro-ordinateur "
(L.R.I.M. 1989)

[BLE 82] : T. BLESER et J. D. FOLEY
" Towards Specifying and Evaluating the Human Factors of User-Computer Interfaces "
(Humans-Factors in Computer Systems, Mars 82, p.309-314)

[BUX 83] : W. BUXTON, M. R. LAMB, D. SHERMAN et K. C. SMITH
" Towards a Comprehensive User Interface Management System"
(Computer Graphics, 17-3, Juillet 83, p. 35-42)

[CAR 85] : L. CARDELLI et R. PIKE
" SQUEAK : A language for Communicating with Mice "
(Computer Graphics, 19-3, Juillet 85, p. 199-204)

[COU 86] : J. COUTAZ
" La construction d'interfaces homme-machine "
(Rapport de Recherche, IMAG, Novembre 86)

[DEM 83] : E. P. DEMPSEY et D. R. OLSEN Jr.
" SYNGRAPH : A Graphical User Interface Generator "
(Computer Graphics, 17-3, Juillet 83, p. 43-50)

[DEN 77] : E. DENERT
" Specification and Design of Dialogue Systems with State Diagrams "
(Proceedings of International Computing Symposium, Liège 1977, p. 417-425)

[END 84] : G. ENDERLE, K. KANSY et G. PFAFF
" GKS : the graphics standart "
(Computer Graphics Programming, SPRINGER VERLAG, Berlin 1984)

[FLE 87] : M. A. FLECCHIA et R. D. BERGERON
" Specifying Complex Dialogs in ALGAE "
(CHI + GI'87, ACM, Avril 87, p. 229-234)

[FOL 74] : J. D. FOLEY et V. L. WALLACE
" The Art of Natural Graphic Man-Machine Conversation "
(IEEE, 62-4, Avril 74, p. 462-471)

[FOL 82] : J. D. FOLEY et A. VAN DAM
" Fundamental of Interactive Computer Graphics "
(Addison Wesley, Reading, 1982)

[GAL 83] : Y. GARDAN et M. LUCAS
" Techniques graphiques interactives et C.A.O. "
(HERMES, 1983)

[GAR 82] : Y. GARDAN
" Eléments méthodologique pour la réalisation de systèmes de CFAO et leur introduction dans l'entreprise "
(Thèse d'Etat, I.N.P.G., Décembre 82)

[GAR 83] : Y. GARDAN
" Systèmes de C.F.A.O. "
(HERMES FRANCE, 1983)

[GAR 86a] : Y. GARDAN
" SACADO : Présentation générale "
(Note interne, LRIM, Août 86)

[GAR 86b] : Y. GARDAN
"La CFAO : introduction, techniques et mise en oeuvre"
(HERMES, 1986)

[GAR 87] : Y. GARDAN et W. TOTINO
" Intérêt d'un modèle externe de visualisation : application aux performances de l'opération d'identification "
(Revue Internationale de CFAO et d'Infographie, HERMES, vol. 2, n. 1, 1987)

[GAR 87b] : Y. GARDAN
" Une nouvelle approche pour la conception de systèmes de C.A.O. : application aux générateurs dans SACADO "
(Rapport de Recherche, LRIM, Mai 87)

[GAR 89a] : Y. GARDAN
" Eléments de CAO - Matériels et logiciels de base "
(HERMES, volume1, 1989)

[GET 87] : J. GETTYS et R.W. SCHEIFLER
" The X Window System "
(ACM, 5-2, Avril 87, p. 79 à 109)

[GRE 85] : M. GREEN
" The University of Alberta User Interface Management System "
(ACM, 19-3, 1985, p. 205-213)

[GRE 86] : M. GREEN
" Survey of Three Dialogue Models "
(ACM, Décembre 86, p. 244-275)

[GUE 82] : S. P. GUEST
" The Use of Software Tools for Dialogue Design "
(Int. J. Man-Machine Studies, 16, 1982, p. 263-285)

[HAN 80] : P. R. HANAU et D. R. LENOROVITZ
" Prototyping and Simulating Tools for User-Computer Dialogue Design "
(Computer Graphics, 14-3, Juillet 80, p. 271-278)

[HEM 86] : B. HEULLUY et G. MICHEL
" Les problèmes de Bases de Données en CFAO : étude bibliographique "
(Revue Internationale de CFAO et d'Infographie, HERMES, vol. 1, n. 1, 1986)

[JAC 85] : R. J. K. JACOB
"A Transition Diagram Language for Visual Programming"
(Computer Magazine, Août 85, p. 51-59)

[JUN 87] : J. P. JUNG
" Le modèle de la maquette 2D de SACADO "
(Note interne, LRIM, Mai 87)

[JUN 89] : J. P. JUNG
" Le modèle géométrique de SACADO "
(Note interne, LRIM, Avril 89)

[KAM 83] : A. KAMRAN et M. B. FELDMAN
" Graphics Programming Independent of Interaction Techniques and Styles "
(Computer Graphics, 17-1, Janvier 83, p. 58-66)

[KAS 82] : D. J. KASIK
" A User Interface Manegement System "
(Computer Graphics, 16-3, Juillet 82, p. 99-106)

[KIE 83] : D. KIERAS et P. G. POLSON
" A Generalized Transition Network Representation for Interactive Systems "
(CHI'83, ACM, 1983, p. 103-106)

[KOL 87] : J. N. KOLOPP
" Les contours dans SACADO "
(Note interne, LRIM, Février 87)

[KOL 88] : J. N. KOLOPP
" Arrangements structurels dans SACADO "
(Note interne, LRIM, Avril 88)

[KOL 89] : J. N. KOLOPP
" Les pièces dans SACADO "
(Note interne, LRIM, Février 89)

[KRZ 86] : T. KRZEWINA
" Une étude comparative de PHIGS et GKS "
(Actes Micad 1986, p. 111 à 124)

[LEV 85] : G. LEVY
" Macintosh et Lisa 2 "
(MAC GRAW-HILL, 1985)

[LIE 85] : H. LIEBERMAN
" There's More to Menu Systems than Meets the Screen "
(Siggraph'85, Comp. Grah. 1985, p. 181-189)

[LRIM 87a] : Y. GARDAN, J.-P. JUNG, J. N. KOLOPP, C. MINICH et W. TOTINO
" La maquette SACADO 2D : architecture générale et dialogue "
(Rapport de Recherche, LRIM, Février 87)

[LRIM 87b] : Y. GARDAN, J.-P. JUNG, J. N. KOLOPP, C. MINICH et W. TOTINO
" SACADO : Guide d'utilisation "
(LRIM 1987)

[LRIM 88] : Y. GARDAN, J.-P. JUNG, J. N. KOLOPP, C. MINICH et W. TOTINO
" Une Approche Nouvelle de la Convivialité dans un Système de C.A.O. : les Principes du Dialogue dans SACADO "
(Actes MICAD 88, HERMES, 21-25 Mars 1988, p. 281-296)

[MAC 88] : Manuel de l'utilisateur du système logiciel Macintosh

[MAR 85] : F. MARQUE
" Opérations booléennes "
(Rapport de Recherche, LRIM, 1985)

[MIC 87] : D. MICHEL
" Etude de l'intersection et du raccordement de deux carreaux de Bézier "
(Rapport de D.E.A., Metz (LRIM) - Compiègne, Septembre 87)

[MIL 77] : L. H. MILLER
" A Study in Man-Machine Interaction "
(Proceedings AFIPS, NCC, 46, 1977, p. 409-421)

[MIN 86] : C. MINICH
" Mise en oeuvre d'extrusion généralisées "
(Rapport de D.E.A., Paris (MDV) - Strasbourg (ULP), Septembre 86)

[MIN 88a] : C. MINICH
" Opérateurs de révolution et d'extrusion : rôle des opérateurs d'Euler "
(Note interne, LRIM, Mars 88)

[MIN 88b] : C. MINICH
" Le modèle volumique de SACADO "
(Note interne, LRIM, Septembre 88)

[MOR 81] : T. P. MORAN
" The Command Language Grammar : a Representation for User Interface of Interactive Computer Systems "
(Int. J. Man-Machine Studies, 15-1, Juillet 81, p. 3-50)

[MSW 88] : Manuel d'utilsation de MS/WINDOWS

[NEW 68] : W. M. NEWMAN
" A System for Interactive Graphical Programming "
(Spring Joint Computer, AFIPS Press, 1968, p. 47-54)

[OLS 84a] : D. R. OLSEN Jr., W. BUXTON, R. EHRICH, D. J. KASIK, J. R. RHYNE et J. SIBERT
" A Context for User Interface Management "
(IEEE Comp. Graph. and Appl., 4-12, Décembre 84, p. 33)

[OLS 84b] : D. R. OLSEN Jr.
" Pushdown Automata for User Interface Management "
(ACM Transactions on Graphics, 3-3, Juillet 84, p. 177-203)

[PAR 69] : L. D. PARNAS
" On the Use of Transition Diagrams in the Design of a User Interactive Computer System "
(Proceedings of the 24-th National ACM Conference, Août 69, p.379-385)

[PEL 84] : M. PELLISSIER et O. LECARME
" La transportabilité du logiciel "
(MASSON, 1984)

[PUC 88] : S. PUCCI
" Coupes demi-espaces - Evaluation d'un arbre CSG "
(Note interne, LRIM, Juin 88)

[PUC 89a] : S. PUCCI
" Les opérateurs d'Euler "
(Note interne, LRIM, Janvier 89)

[PUC 89b] : S. PUCCI et Y. GARDAN
" Utilisation des octrees pour la conversion d'un modèle CSG à un modèle par les frontières "
(Rapport de Recherche, LRIM, Avril 89)

[PUC 89c] : S. PUCCI
" Structure hiérarchique de localisation pour l'évaluation des frontières d'un arbre de construction "
(Thèse de Doctorat de l'Université de Strasbourg (préparée au LRIM), à paraître)

[REI 81] : P. REISNER
" Formal Grammar and Human Factors Design of an Interactive Graphics System "
(IEEE Trans. Softw. Eng., SE 7-2, Mars 81, p. 229-240)

[REI 83] : P. REISNER
" Analytic Tools for Human Factors of Software "
(IBM Research Laboratory Report RJ 3803, 1983)

[ROA 82] : J. ROACH, R. H. HARTSON, W. R. EHRICH, T. YUNTEN et D. H. JOHNSON
" DMS : A Comprehensive System for Managing Human-Computer Dialogue "
(Human-Factors in Computer Systems, 1982, p. 102-105)

[ROM 87] : V. ROMAGNOLI et R. SCABBIA
" Man-Machine Interaction : a Different Way to Use Menus "
(Actes MICAD 87, HERMES, 23-27 Février 1987, p. 27-38)

[ROS 82] : D. S. H. ROSENTHAL
" The Detailed Semantics of Graphics Input Devices "
(Computer Graphics, 16-3, Juillet 82, p. 33-38)

[SAH 87] : M. SAHNOUNE
" Conception d'un logiciel de modélisation d'hélices bipales d' U.L.M. "
(Rapport de D.E.A., Metz (LRIM) - Strasbourg (ULP), Septembre 87)

[SCH 85] : A. J. SCHULERT, T. G. ROGERS et J. A. HAMILTON
" ADM : A Dialog Manager "
(CHI'85, ACM, Avril 85, p. 177-183)

[SIB 86] : J. L. SIBERT, W. D. HURLEY et T. W. BLESER
" An Object-Oriented User Interface Management System"
(ACM, 20-4, 1986, p. 259-268)

[SLO 89a] : I. SLOMIANY
" Les Pièces Paramétrées dans SACADO"
(Note interne, LRIM, Mars 89)

[SLO 89b] : I. SLOMIANY
" Un langage interprété spécifique à la description de pièces paramétrées en C.A.O."
(Mémoire C.N.A.M., LRIM, à paraître [Octobre 89])

[TOT 86] : W. TOTINO
" Bases d'un système de CAO ouvert : application au dialogue "
(Rapport de D.E.A., Metz (LRIM) - Strasbourg (ULP), Septembre 86)

[TOT 87] : W. TOTINO
"Le point sur le dialogue et les problèmes rencontrés"
(Note interne, LRIM, Mai 87)

[TOT 88] : W. TOTINO
" Multi-fenêtrage et Multi-clôturage "
(Note interne, LRIM, Avril 88)

[TOT 89] : W. TOTINO et Y. GARDAN
" Interface homme-machine dans un système graphique "
(Rapport de Recherche, LRIM, Avril 89)

[TRE 77] : S. TREU
" A Framework of Characteristics Applicable to Graphical User-Computer Interaction "
(User Oriented Design of Interactive Graphics Systems, 1977, p. 61-71)

[VAN 83] : J. VAN DEN BOS et P. H. HARTEL
" Input-Output Tools : A Language Facility for Interactive and Real-Time Systems "
(IEEE Trans. Softw. Eng., SE 9-3, Mai 83, p. 247-259)

Résumé :

La modélisation de l'interface homme-machine dans un système graphique interactif reste un problème ouvert dans la mesure où il est encore mal défini. Dans ce cadre, nous situons le sujet par rapport au projet SACADO du LRIM. Après quoi, nous tentons de définir et d'élaborer une architecture logicielle en proposant un modèle de dialogue qui s'articule autour d'une structure générique dynamique (liste de menus et mécanismes associés). Une extension nous conduit à l'introduction du concept d'*"interaction unique"*, prenant en compte des notions de *"compatibilité"* des intentions de l'utilisateur quel que soit le contexte. Les principes défendus ont permis la mise en oeuvre d'un système opérationnel qui se caractérise par l'unification de la gestion du dialogue (pour l'utilisateur, le concepteur et le programmeur), ainsi que par la dualité qui existe entre le gestionnaire de dialogue et une application.

Mots clés :

C.A.O. (Conception Assitée par Ordinateur), Concepteur, Dialogue, Gestionnaire de Dialogue, Interaction, Interface Homme-Machine, Modélisation, Modèle Générique, Opérateur, Progammeur.