MASTER EN SCIENCES INFORMATIQUES

Élaboration d'une boîte à outils d'aide à la réalisation d'interfaces pour personnes handicapées

Baudrenghien, Stéphanie; Demo, Rudy

Award date:
1992

Awarding institution:
Universite de Namur

Année Académique 1991-1992

"Elaboration d'une boîte à outils d'aide à la réalisation d'interfaces pour personnes handicapées"

Stéphanie Baudrenghien **Rudy Demo**

Promoteur : **Mme Monique Noirhomme-Fraiture**

Mémoire présenté en vue de
l'obtention du grade de Licencié
et Maître en informatique

« *Toi dont le vêtement est si propre, dont la métaphysique est si belle, toi dont quelques carrés de papiers ont acheté le respect de ta société, toi qui sais si bien dire* ***Moi****... pourquoi as-tu appris la maladie? Pourquoi connais-tu comment souffrir et vieillir? Quelle est la force qui te fais détourner la tête lorsqu'un homme meurt?* »

Extrait de "Le Voyage à Shambhalla : Un pélerinage vers Soi" d'Anne et Daniel Meurois-Givaudan, éd. Arista

Résumé

Ce travail présente une application particulière de l'outil informatique dans le domaine du handicap. Il s'agit d'une boîte à outils d'aide à la réalisation d'interfaces pour personnes handicapées. Les outils doivent être réalisés sous Windows.

Dans ce document, nous exposons l'évolution du travail de l'énoncé initial du problème au codage des outils.

Dans ce cadre et afin de situer notre travail dans son contexte, nous effectuons d'abord un modeste état de l'art des logiciels existant dans le domaine de l'aide à la réalisation d'interfaces sous l'environnement qui nous occupe.

Nous expliquons également les caractéristiques des outils offerts et la raison de nos choix. Bien que ce travail informatique ne fasse pas appel à des notions de psychologie de la personne handicapée, il reconnaît la nature des difficultés de réalisation d'une interface pour ce type de population. En conséquence, le choix des outils à offrir a été effectué après analyse d'autres mémoires réalisés précédemment dans le domaine du handicap.

En outre, OBLOG, une méthode orientée objet, est présentée et utilisée pour la conception des outils.

Après avoir disserté sur l'environnement sous lequel nous avons travaillé et éclairé le lecteur sur les problèmes rencontrés dans ce type d'environnement, nous avons décidé de nous pencher sur les perspectives offertes par la boîte à outils réalisée.

En effet, les outils sont destinés à être intégrés dans un logiciel d'élaboration d'interfaces. Ce logiciel reste à développer mais nous en avons dégagé les principales caractéristiques.

Abstract

This document describes a specific computer application for disabled persons. It consists of a software toolbox to help in developing interfaces for said type of users. The tools are to be developed under Windows.

In the present document, we describe our activity as it built up from the initial title to the code implementation.

In this respect and in order to define our work within its context, we first present a short state of the art of the existing softwares dedicated to the interface implementation under the specified environment.

We explain the characteristics of the tools we have chosen and why they were chosen. Although the present software development activity does not address the disabled person psychology, it describes the actual difficulty to implement an interface for this type of users. Consequently, the tools have been chosen after a survey of other schoolyear projects formerly written about the same topic.

In addition, OBLOG, an object oriented method, is presented and applied for the tools development.

After some considerations about the environment and explanations about the difficulties relating to that kind of environment, we decided to explain the prospects of the implemented toolbox.

The intent is to integrate the tools into an interface development software. This software is still to be developed but we have prepared its main characteristics.

Remerciements

C'est pour nous un plaisir d'exprimer nos remerciements sincères à toutes les personnes ayant collaboré, de près ou de loin, à la réalisation de ce mémoire.

Nous remercions plus particulièrement :

Madame le Professeur M. Noirhomme, qui a accepté de parrainer ce travail et nous a prodigué aide, encouragements et conseils précieux tout au long de la présente élaboration.

Monsieur le Professeur E. Dubois, pour les explications ou éclaircissements de certains concepts orientés objet.

Messieurs B. Decuyper, B. Sacré et J. Vanderdonckt, pour leur aide technique et la documentation qu'ils nous ont fournie.

La Région Wallonne, pour son support financier nous ayant permis d'effectuer notre stage en Angleterre.

Monsieur le Professeur P. Golder, Monsieur D. Bigaux et Mademoiselle J. Reymond, qui nous ont chaleureusement accueillis à Birmingham.

Monsieur C. Baudrenghien et Madame F. Rousseau pour leur aide et leur support dans notre travail.

Plan du mémoire

INTRODUCTION

Chapitre 1 : Historique et généralités

Chapitre 2 : Etat de l'art sommaire des divers outils d'aide à la conception d'interfaces graphiques

Chapitre 3: Choix d'une méthode de développement orientée objet

Chapitre 4 : Choix et analyse des objets interactifs

Chapitre 5 : Spécification d'exercices sur base de nos objets graphiques

Chapitre 6 : Windows, sa structure, son fonctionnement et son influence sur notre travail

Chapitre 7 : Vers un logiciel d'élaboration d'interfaces

Introduction

L'informatique, en se développant, trouve des applications dans des secteurs de plus en plus étendus de notre activité.

L'Institut d'Informatique des Facultés Notre-Dame de la Paix propose, depuis quelques années déjà, des sujets de mémoires consacrés aux applications pour personnes handicapées sous la direction de Madame le Professeur M. Noirhomme ou de Monsieur le Professeur J. Ramaekers.

Nous avons choisi l'un de ces sujets, non seulement en raison de l'aspect humain et désintéressé de l'entreprise, mais aussi du défi technique que présente un tel sujet.

D'abord, il nous fallait choisir des objectifs à la mesure du temps et des moyens à notre disposition. Le titre proposé : "Elaboration d'une boîte à outils d'aide à la réalisation d'interfaces pour personnes handicapées" implique l'idée d'une collection plus ou moins importante d'outils informatiques à développer. Nous en avons développé autant que possible. Les outils sont destinés à un logiciel d'élaboration d'interfaces. Ce logiciel reste à développer mais nous en avons dégagé les principales caractéristiques.

Le logiciel d'élaboration d'interfaces, s'il est bien conçu, permettra à des éducateurs d'élaborer eux-mêmes des applications simples et adaptées aux personnes handicapées dont ils ont la charge.

De nouveaux outils pourront toujours enrichir les ressources disponibles pour la création d'interfaces de plus en plus spécialisées.

L'enseignement en général est une affaire de spécialistes. L'enseignement aux personnes handicapées sur ordinateur ne peut se concevoir qu'avec l'aide d'éducateurs expérimentés.

Bien que ce travail informatique ne fasse pas appel à des notions de psychologie de la personne handicapée, il reconnaît la nature des difficultés qui existent dans l'établissement d'une communication. L'éducateur doit, en effet, pouvoir choisir la difficulté d'un exercice en fonction de la capacité de compréhension de son élève, alors que l'informaticien doit, grâce aux moyens logiciels et matériels, permettre un accès facile à l'ordinateur.

En conséquence, pour déterminer les outils à réaliser, nous avons lu les mémoires traitant du sujet "informatique et handicap" et nous avons tenté d'en tirer les traits communs.

Dans les premier et deuxième chapitres, nous donnerons les bases du travail et éclairerons le lecteur sur la pertinence et l'originalité de cette élaboration.

Le troisième chapitre exposera la méthode utilisée pour concevoir les outils.

Nous procéderons ensuite, dans le chapitre suivant, à l'explication du choix des outils et à leur analyse selon la méthode exposée dans le troisième chapitre.

Le chapitre cinq constituera une illustration par l'exemple de la pertinence des outils pour la conception d'interfaces destinées à des personnes handicapées.

Le chapitre six présentera l'environnement utilisé pour l'implémentation des outils et les problèmes que nous avons rencontrés face à ce dernier.

Nous terminerons le travail en dégageant les caractéristiques principales que devra présenter le logiciel d'élaboration d'interfaces à construire sur base de nos outils.

Chapitre 1: Historique et généralités

Introduction

Ce chapitre a pour objet de définir les bases de notre travail. L'historique présenté éclairera le lecteur sur le cheminement conduisant à la définition du travail à effectuer tandis que les généralités poseront les jalons nécessaires à la compréhension de ce dernier.

1.1.Historique

Nous allons, dans la présente section, indiquer l'évolution des idées sous-jacentes à l'intitulé et les différentes discussions ayant permis de le clarifier.

1.1.1. Les bases

Pour réaliser ce travail, nous avons d'abord reçu un pré-cahier des charges d'un centre pour personnes handicapées situé à Kain : "Le Trèfle". Ce document (annexe 1), réalisé par deux éducatrices du centre, Mesdames Quintin et Vandiependael, est un ensemble de propositions d'exercices simples d'acquisition de connaissances que les éducatrices aimeraient voir réaliser.

A l'Institut d'Informatique des Facultés, d'autres mémoires ont été réalisés sur des sujets similaires. Au vu de ces logiciels, force est de constater qu'ils sont tous des outils spécifiques qui, même s'ils sont paramétrables, sont utilisables par une population restreinte et pour un but bien défini. Donnons l'exemple du logiciel "Compte" [Cher, 90]. Ce dernier est utilisable par des personnes ayant la capacité d'utiliser un ordinateur, ayant conscience qu'il faut dépenser de l'argent pour acheter quelque chose, ayant la capacité de reconnaître assez bien les pièces et les billets et ayant une certaine somme à leur disposition. De plus, il n'est utilisable que pour la gestion de cet argent.

Toute l'originalité du présent mémoire réside dans son ambition de concevoir un logiciel plus général. Un tel logiciel offrirait un ensemble d'outils facilitant la réalisation d'exercices spécifiques pour personnes handicapées.

Le travail qui nous est attribué est le suivant :

- analyse des travaux précédents pour déterminer les éléments communs, leurs caractéristiques et déduction des outils à offrir,

- réalisation des outils,

- réalisation, grâce aux outils, de quelques exercices proposés par le Trèfle.

Le travail doit être réalisé sur PC sous Windows.

1.1.2. Le stage en Angleterre

Le stage que nous avons effectué en Angleterre (rapport en annexe 2) nous a permis d'examiner des logiciels existant dans le domaine qui nous intéresse et de rencontrer des concepteurs et utilisateurs de ces logiciels. Nous avons visité des centres, des institutions spécialisées et des universités dans les villes de Loughborough, Oxford et Birmingham.

La visite de l'"A.C.E. centre" d'Oxford fut particulièrement intéressante. Elle nous a permis d'observer les réactions de personnes possédant un handicap face à différents logiciels et de participer aux séances d'évaluation qui suivirent l'observation. En outre, nous avons pu expérimenter un logiciel permettant de réaliser des exercices spécifiques pour personnes handicapées (le logiciel PLOCKA).

Quant aux visites des différents centres et de l'université de Birmingham, elles nous ont permis de nous rendre compte des domaines nécessitant des programmes d'apprentissage et des points importants à prendre en considération lors de la conception d'une application pour personnes handicapées.

Le stage nous a aidé à prendre conscience du potentiel que représente l'outil informatique pour l'acquisition de connaissances par des personnes ayant des déficiences physiques et/ou mentales. Mais ce potentiel ne sera correctement exploité que si l'outil, aussi bien matériel que logiciel, est parfaitement adapté à l'utilisateur. L'adaptation matérielle peut être réalisée par les nombreux périphériques existants (souris, écran tactile, interrupteurs,...) tandis que l'adaptation au niveau du logiciel sera plus complexe. En effet, elle est un compromis entre de nombreux facteurs. On peut, dès à présent, mettre en lumière que le type d'information à présenter à l'écran doit être pertinent pour l'utilisateur (on ne va pas présenter des informations textuelles à une personne ne sachant pas lire). C'est pourquoi nous avons pu constater l'abondance de logiciels utilisant des interfaces graphiques, accessibles à un plus grand nombre d'utilisateurs.

Différentes constatations s'imposent à l'analyse des logiciels expérimentés durant le stage :

- ils sont souvent fort spécifiques. Ils ne conviennent donc qu'à un type d'utilisateurs ayant un profil bien déterminé,

- ils sont souvent des logiciels d'apprentissage destinés à des enfants et repris, faute de mieux, pour les personnes handicapées. Ce type de logiciel démotive souvent les personnes adultes qui sont frustrées par leur caractère enfantin,

- ils nécessitent parfois des manipulations par l'éducateur durant l'exercice, ce qui distrait inévitablement l'utilisateur,

- ils ne nécessitent que peu de traitement mais donnent beaucoup d'importance à l'interface (presqu'exclusivement graphique).

- il existe une demande importante de la part des éducateurs pour des logiciels permettant l'apprentissage de scènes de la vie quotidienne (prendre un ticket de bus, aller à la piscine, utiliser un distributeur automatique de billets...).

La diversité des demandes en matière de logiciels nous a convaincus du bien fondé de la proposition de conception d'une boîte à outils faite par Madame M. Noirhomme.

De notre côté, nous pensions alors réaliser un logiciel offrant différents outils aux éducateurs et leur permettant d'élaborer simplement différents exercices graphiques adaptés à l'utilisateur particulier auquel ils sont destinés.

1.1.3. Les discussions qui suivirent le stage

Au retour du stage, nous avons exposé notre idée à Madame le Professeur M. Noirhomme. L'idée lui sembla bonne mais bien trop ambitieuse en raison du temps qui nous était imparti. En effet, un tel logiciel serait constitué de l'ensemble exhaustif des outils nécessaires à la création d'exercices pour personnes handicapées. Les outils seraient intégrés dans un logiciel plus général destiné aux éducateurs et leur permettant par un ensemble de manipulations simples de créer des exercices "à la carte" pour la population concernée. Ce travail était bien trop important pour deux personnes travaillant pendant un an seulement. Il nécessitait, non seulement l'analyse et la réalisation des outils, mais aussi l'analyse de leurs interactions et de l'interface du logiciel. En effet, les éducateurs n'étant pas des experts en informatique, il fallait que le logiciel leur soit adapté.

En conséquence, les objectifs du mémoire ont été recadrés et le titre de celui-ci a été fixé. Le titre en est : "Elaboration d'une boîte à outils d'aide à la réalisation d'interfaces pour personnes handicapées" et le travail consiste à présent en :

- l'analyse des mémoires précédemment réalisés pour déterminer un ensemble d'outils à offrir (tout en restant conscients que l'ensemble des outils déterminés n'est pas complet. Il sera complété par d'autres étudiants lors de mémoires ultérieurs),

- la réalisation de quelques-uns d'entre eux,

- une première approche du futur logiciel permettant la réalisation d'interfaces graphiques pour la personne handicapée par l'utilisation des outils que nous avons créés,

- la réalisation de quelques exercices pour le Trèfle, si le temps nous le permet.

Une part importante de notre temps a été consacrée à l'apprentissage de Windows et du langage C nécessaire à la création de routines sous Windows. Nous n'avons pas choisi d'étudier le C++ en raison du manque de temps, bien qu'il soit mieux adapté à cet usage.

1.1.4.La rencontre avec les éducatrices du Trèfle

Une rencontre avec les éducatrices du Trèfle nous a permis de raffiner le pré-cahier des charges reçu. En supplément des précisions apportées, les éducatrices nous ont proposé un feed-back lorsque les quelques exercices réalisés leur seraient remis. En effet, elles se proposent, après les avoir expérimentés, de nous communiquer la pertinence du choix des outils. Ceci permettra éventuellement à l'occasion de mémoires ultérieurs, de modifier ou d'ajouter des outils avec pour objectif final la conception d'un logiciel permettant la réalisation d'interfaces graphiques pour personnes handicapées.

1.2.Généralités

Nous avons indiqué précédemment que l'outil informatique est utilisé pour aider les personnes handicapées à acquérir des connaissances mais que cette aide n'est possible que si l'outil est parfaitement adapté à l'utilisateur. Nous présenterons, ci-dessous, les problèmes liés à l'acquisition de connaissances par des personnes présentant un handicap. Nous déterminerons ensuite les problèmes liés à l'adaptation de l'outil à cet utilisateur et les choix effectués pour réaliser au mieux l'adaptation.

1.2.1.Problèmes d'acquisition de connaissances

Cette section présentera une définition de ce qu'est la connaissance et les moyens d'acquérir des connaissances par des personnes dites normales. Nous dégagerons ensuite les problèmes liés à l'acquisition de connaissances par des personnes handicapées et les moyens de les aider dans le processus d'acquisition.

A. Définition de la connaissance

Selon le Petit Robert, la ***connaissance,*** c'est "ce qui est connu; ce que l'on sait, pour l'avoir appris, ce que l'on a dans l'esprit en tant qu'objet de pensée analysé". La connaissance serait donc un **état de l'esprit**.

L'***acquisition de connaissances***, quant à elle, serait un processus qui augmente la quantité et l'efficience des composants de cet état.

L'acquisition de connaissances est réalisée à l'aide de différents supports (livres, ordinateurs, conversations...). Quel que soit le support employé, elle nécessitera toujours l'intervention du processeur humain.

Selon Card, Moran et Newell, "le modèle du processeur humain est un ensemble de mémoires et de processus interconnectés". Ces processus sont traités par trois processeurs : le processeur perceptif, le processeur cognitif et le processeur moteur [Card,83].

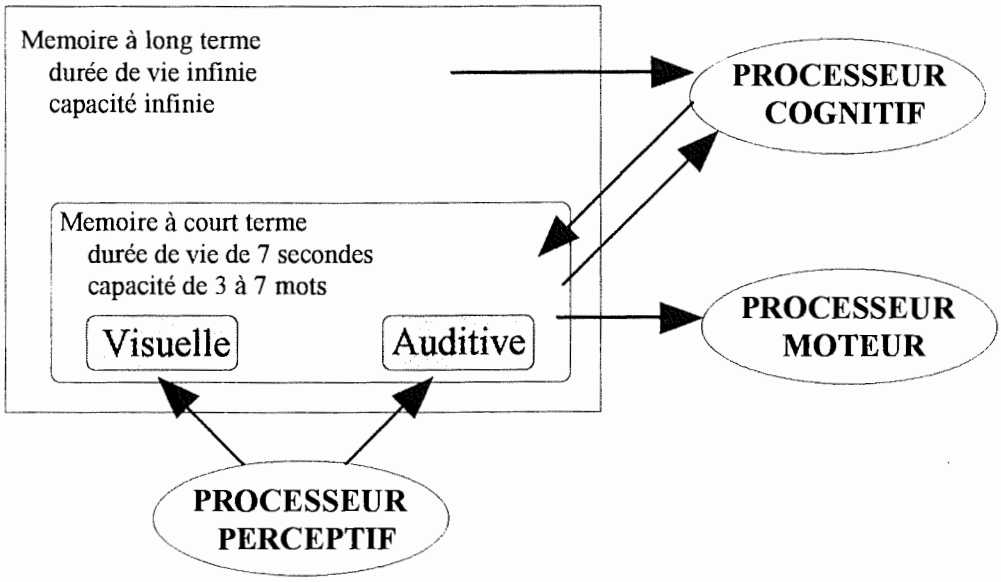


figure 1.1 : Le modèle du processeur humain
selon Card, Moran et Newell

L'acquisition de connaissances peut être le résultat d'exécution de tâches ou d'exercices. La réalisation de toute tâche nécessitera le traitement de cette dernière par le biais des processeurs.

"Une tâche est définie par un but, des préconditions et des opérations ou des méthodes. Le domaine d'une tâche est structuré en un ensemble d'objets caractérisés par leurs propriétés et les opérations sur ces objets"[Boy,91].

Notons que si la tâche exécutée pour acquérir des connaissances est réalisée au moyen d'un support, par exemple un livre, il faut que ce support soit compréhensible pour l'utilisateur. Pour notre exemple, une personne ne comprenant pas le français sera incapable de mener à bien une tâche dont l'énoncé est rédigé dans un livre en français.

Si le support utilisé est l'ordinateur, la bonne exécution des tâches n'est rendue possible que grâce à la qualité de l'***interface,*** "système informatique utilisé par une personne pour réaliser une tâche à accomplir à l'aide d'un ensemble de moyens informatiques" [Boda,90].

Nous pouvons donc fort bien concevoir qu'une mauvaise interface entraînera certainement des problèmes au niveau de la réalisation de la tâche. Ceux-ci pourraient être induits par :

- un message non perçu,

- un message mal compris ou mal interprété. L'incompréhension peut être une conséquence d'une réception influencée par le référentiel de l'utilisateur. En effet, ce dernier comprend ce qu'il s'attend à recevoir en fonction de son savoir-faire, de son expérience, de sa culture...

- un moyen d'interaction inadapté. Comment réaliser correctement une tâche si on ne peut interagir avec le système de façon adéquate?

B. Le domaine du handicap

S'il n'est pas facile d'adapter l'outil informatique pour permettre l'acquisition de connaissances par une population d'utilisateurs dits normaux, il sera malheureusement encore bien plus difficile d'aider les personnes présentant un handicap à réaliser cette acquisition. En effet, ces dernières présentent déjà des lacunes au niveau du processeur

humain. Que dire de leur référentiel culturel? De leur savoir-faire? De leurs réactions face à l'ordinateur?

Cependant, nous voudrions y arriver en nous basant sur l'**attrait de l'ordinateur** et grâce à une **utilisation répétée** de ce dernier.

Le support informatique est attrayant. La personne handicapée, lorsqu' il lui est donné la possibilité de travailler sur un ordinateur, se sent mise en valeur. En effet, elle est habituée à travailler avec différents supports classiques (livres, jeux,...) mais toujours avec un éducateur pour l'aider. Le support informatique, quant à lui, peut être utilisé indépendamment de l'éducateur. Cette neutralité de l'ordinateur par rapport à l'éducateur est une source de motivation pour la personne présentant un handicap.

D'autre part, l'ordinateur est un enseignant particulièrement patient et inlassable. Il permet à l'utilisateur de travailler seul, à son propre rythme et lui évite d'avoir à montrer ses erreurs aux autres.

De plus, l'utilisation de l'ordinateur renverse le rapport enseignant-enseigné. En effet, l'ordinateur, contrairement à d'autres supports, permet à l'enseigné de devenir actif (ou plutôt réactif, comme nous le verrons plus loin) face à l'enseignement. La personne présentant un handicap a alors le sentiment de pouvoir agir de sa propre initiative sans la crainte souvent insupportable d'être réprimandé ou même simplement observé lorsqu'elle agit mal.

Le caractère ludique des exercices présentés lui permettra en outre de surmonter la peur d'agir, si souvent présente chez ce type de personne.

Si ces différents éléments sont autant de facteurs rendant l'outil informatique attrayant pour la personne handicapée, ils ne pourront assurer seuls la motivation nécessaire à l'acquisition de connaissances. En effet, pour une population pour laquelle les sujets de désintérêt sont bien plus nombreux que les sujets d'intérêt, la motivation ne pourra être obtenue et maintenue que par une interface parfaitement adaptée.

L'utilisation répétée est essentielle. C'est grâce à elle que l'utilisateur acquerra des connaissances. Cependant, si les performances de la personne handicapée lors de la réalisation d'exercices augmentent avec le nombre d'utilisations, à partir d'un certain optimum, elles commenceront à diminuer. Ceci s'explique par la perte de motivation; la personne se lasse des exercices qui lui sont présentés. Il faudra alors proposer de nouveaux exercices un peu plus difficiles pour permettre à nouveau la motivation qui, si elle est associée à une utilisation répétée du système, entraînera une acquisition de connaissances. Nous pouvons illustrer le raisonnement par le schéma de la figure 1.2..

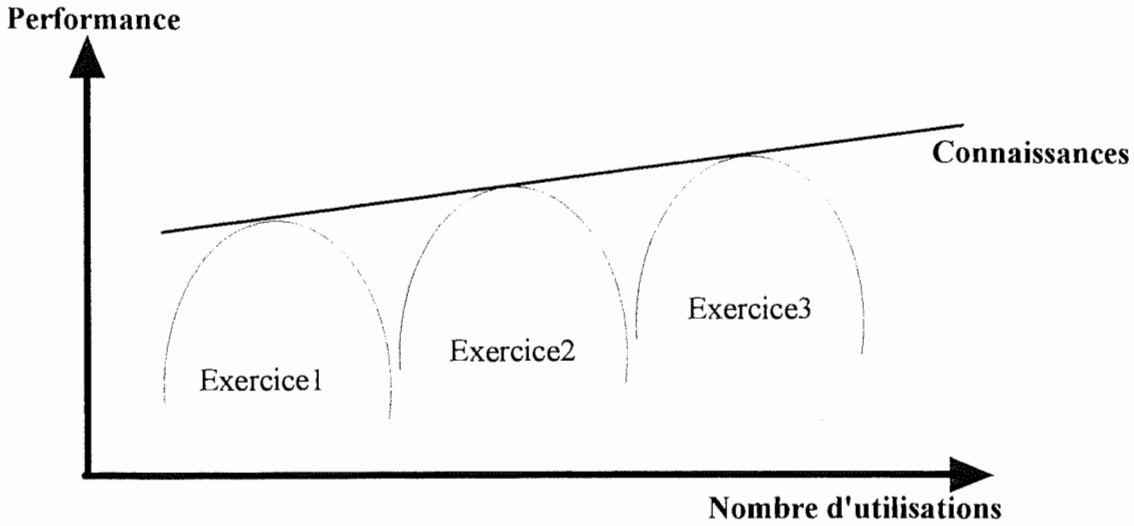


Figure 1.2 : Acquisition de connaissances

Les courbes de performance peuvent cependant présenter des imperfections. Si on observe une brusque régression, cela peut s'expliquer par le fait que les personnes handicapées mentales ont des comportements assez différents d'un jour à l'autre et n'est donc pas toujours attribuable à une perte de motivation. Nous pouvons également être plus en forme un jour qu'un autre... Même si chez les personnes handicapées mentales les différences sont plus accentuées, une régression doit parfois être vue comme un accident.

1.2.2. Problème de la connaissance des acteurs

Cette section traitera des mécanismes mis en oeuvre par tout utilisateur pour réaliser une tâche et de la nécessité pour le concepteur de pouvoir les appréhender en vue d'adapter l'outil informatique à l'utilisateur. Nous expliquerons ensuite pourquoi, dans un tel contexte, nous avons décidé de recourir à une personne intermédiaire pour bien réaliser l'adaptation.

A. Mécanismes mis en oeuvre pour réaliser la tâche

Pour construire un système permettant de réaliser une tâche et qui soit adapté à un utilisateur, il faut d'abord comprendre les mécanismes mis en oeuvre lors de l'exécution de cette tâche. Bien qu'une théorie de la réalisation de la tâche n'existe pas à ce jour (en raison, notamment de la méconnaissance du fonctionnement du cerveau humain), nous pouvons dégager différentes étapes intervenant dans sa réalisation.

Les buts de l'utilisateur sont exprimés en termes psychologiques alors que les mécanismes du système et les états de celui-ci sont exprimés en termes physiques. Norman [Norm, 86] représente cette divergence comme deux gouffres au-dessus desquels il faut construire un pont: le gouffre de l'exécution et le gouffre de l'évaluation. Ceci est illustré par la figure 1.3.

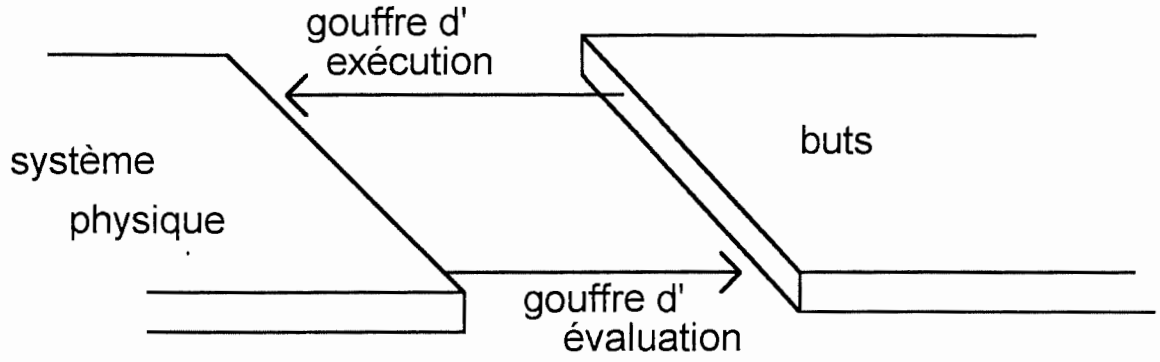


Figure 1.3 : Les deux gouffres

Le concepteur peut combler les gouffres en construisant une interface adaptée aux besoins psychologiques de l'utilisateur. L'utilisateur, lui, peut combler les gouffres en établissant des plans, des séquences d'actions et des interprétations de la tâche à accomplir plus proches de la description requise par le système.

Les étapes mises en oeuvre par un utilisateur actif au cours de la réalisation de sa tâche sont indiquées ci-dessous. Une représentation graphique en est donnée à la figure 1.4.. Certaines d'entre elles peuvent être répétées, d'autres éludées.

- établir le but,
- former l'intention,
- spécifier l'action,
- exécuter l'action,
- percevoir l'état du système,
- interpréter l'état du système,
- évaluer l'état du système par rapport aux buts et aux intentions.

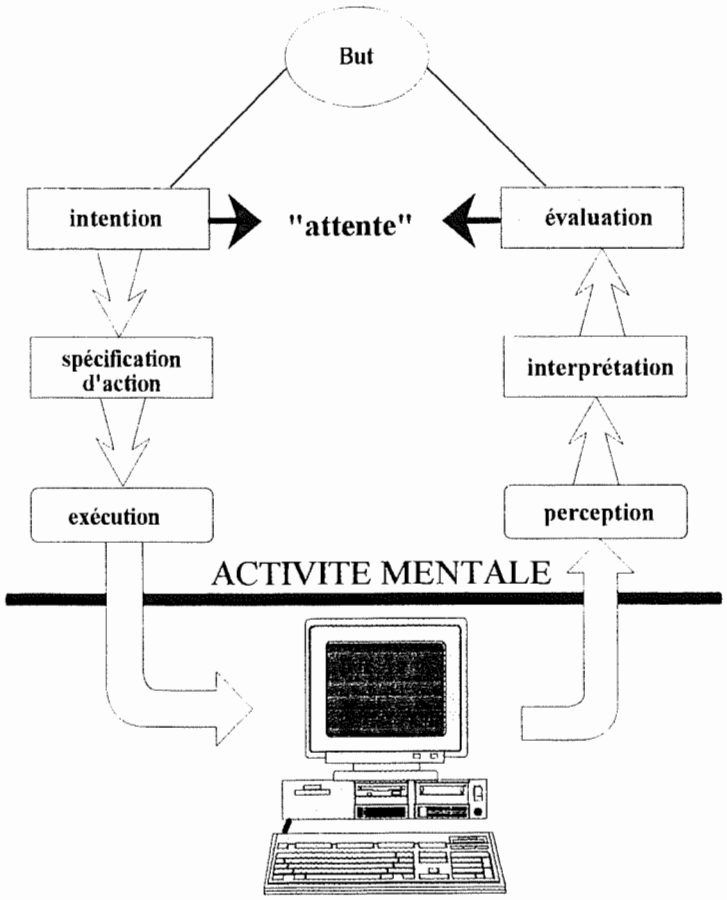


Figure 1.4. : Etapes mises en oeuvre par un utilisateur lors de la réalisation d'une tâche

Cependant, dans certaines situations, l'utilisateur est réactif. Il répond alors à des événements plus qu'il n'établit des buts ou ne forme des intentions a priori. Mais en réponse à ces événements, il est, de manière générale, amené à prendre des décisions, à réagir au système, en établissant des buts et en formant des intentions.

B. Les modèles conceptuels

Tout homme construit un modèle mental interne de ce qui l'entoure. Ce modèle évolue naturellement par les interactions avec le monde ou le système considéré. Il varie fortement en fonction de la nature des interactions, du niveau initial des connaissances de l'individu et de son niveau de compréhension. Il n'est jamais complet ni précis mais il guide fortement le comportement humain.

Dans un système, deux intervenants construisent chacun un modèle mental; le modèle du concepteur et le modèle de l'utilisateur. Un autre concept est celui de l'image résultant de la structure physique du système construit. Le modèle du concepteur est un

modèle conceptuel du système à construire. Cette conceptualisation est basée sur la tâche de l'utilisateur, ses exigences et ses capacités. Elle doit donc tenir compte du contexte de l'utilisateur, de son expérience et de la puissance de ses mécanismes manipulant l'information. Le modèle de l'utilisateur, quant à lui, est construit à partir de l'image qu'il se fait du système. Une représentation de ces concepts est donnée à la figure 1.5.

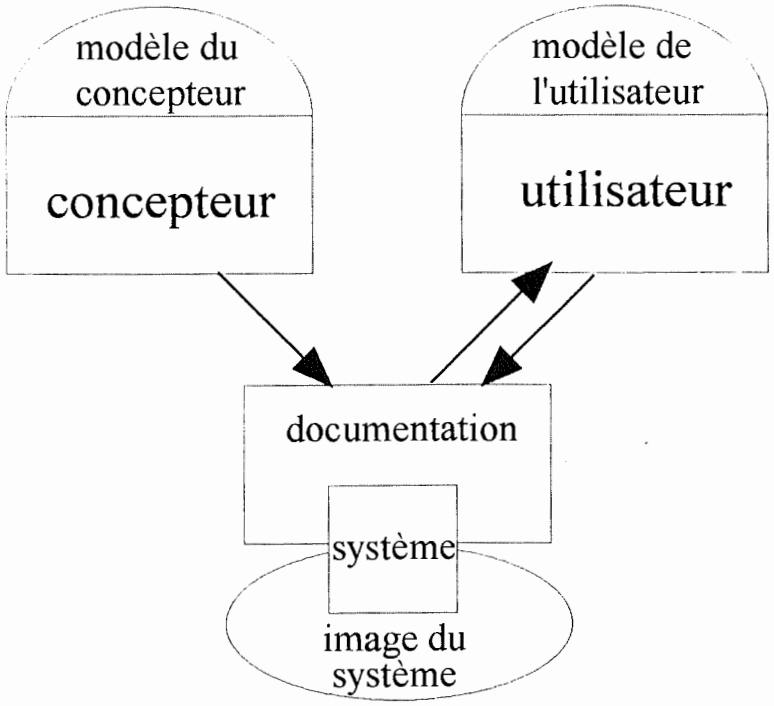


Figure 1.5 : Les modèles conceptuels

Comme l'utilisateur de notre logiciel possède un faible niveau de connaissances, il aura des difficultés à reproduire l'image d'un modèle conceptuel complexe. Le concepteur, désireux de rendre compatible son modèle et celui de l'utilisateur, doit donc s'efforcer de donner une image du système qui soit explicite, intelligible et logique [Norm, 86].

C. La conception d'une bonne interface

Pour concevoir un outil convivial, c'est à dire un outil facile et agréable à utiliser pour des utilisateurs de divers niveaux d'expérience, il faut, comme indiqué précédemment, réussir à proposer une image du système rendant compatible le modèle du concepteur et l'image qu'en perçoit l'utilisateur. Le concepteur engagé dans cette démarche devra faire des choix face aux dilemmes en présence :

- **dilemme de l'intelligence de l'outil**: si un outil est trop intelligent, l'utilisateur deviendra un observateur passif et démotivé. D'autre part, un outil trop peu intelligent

provoquera des problèmes dans le chef de l'utilisateur lorsqu'il devra combler le gouffre d'exécution [Norm, 86].

- **dilemme de la spécialisation**: l'utilisateur désire un outil de haut niveau dont les composants sont proches de la tâche qu'il s'est fixé. Si le concepteur réalise un tel outil, il court le risque de voir les possibilités d'utilisation de son outil extrêmement réduites. De même, s'il conçoit un outil trop général, il démotivera l'utilisateur inexpérimenté.

- **dilemme de la quantité d'information**: une augmentation de la quantité d'information présentée à l'écran peut, si elle n'est pas trop importante, augmenter le niveau de compréhension du système. Cependant, elle provoquera aussi une diminution de l'espace mémoire disponible et du temps de réponse du système.

- **dilemme du niveau de connaissance de l'utilisateur**: si l'outil est destiné à des utilisateurs possédant un faible niveau de connaissance, l'interface devra être beaucoup plus riche et mieux adaptée quant à son contenu informatif, que s'il est destiné à des utilisateurs "experts".

Bien que ne prétendant pas être exhaustifs dans la présente section, nous pouvons remarquer que la conception d'une interface est un compromis entre un nombre important de facteurs. Ces facteurs étant fortement interconnectés, une prise de position concernant certains facteurs aura des implications sur d'autres facteurs.

D. La décision de recourir à une personne intermédiaire

Comme nous l'avons mentionné, l'utilisateur des exercices réalisés est une personne ayant une déficience mentale et/ou physique. Ces personnes possédant en général un faible niveau de connaissance et ne possédant pas les capacités de décision et de discernement des personnes dites normales, leurs comportements face à la machine seront souvent réactifs.

Il faudra, d'une part, concevoir un outil spécialisé et puissant qui propose une quantité d'informations à l'écran nécessaire et suffisante à la compréhension de la tâche. Et, d'autre part, il faudra veiller à concevoir un outil qui ne soit pas trop spécialisé. En effet, les personnes présentant une déficience physique et/ou mentale forment une population non homogène. Une personne n'est pas l'autre. Certaines peuvent être distraites par les informations explicatives présentes à l'écran alors que d'autres n'agiront pas avant d'avoir obtenu l'information nécessaire, certaines savent lire et d'autres pas...

"La conception d'une interface pour ce type de population est toujours un art, une forme d'imitation artistique"[laur, 86], que ce soit en raison des lacunes au niveau de la connaissance des mécanismes du cerveau humain ou en raison des difficultés rencontrées face aux méthodes nécessaires à l'ergonomie de conception. En effet, comment mettre en pratique des méthodes participatives avec des personnes incapables d'exprimer elles-mêmes leurs propres besoins? Comment déterminer l'interface permettant de réduire au maximum les efforts cognitifs à mettre en oeuvre pour réaliser la tâche, l'interface adaptée aux besoins psychologiques de l'utilisateur alors que celui-ci a une rationalité, une logique différente de la nôtre? Et pourtant, ce type d'utilisateurs a, plus que d'autres, besoin d'une interface qui lui soit adaptée. Comment résoudre le problème?

Nous aurons recours à une personne intermédiaire.

Cet intermédiaire est un éducateur. En effet, l'éducateur est un expert dans le domaine du handicap. Moyennant un minimum de connaissances dans le domaine informatique, il doit être capable d'utiliser le logiciel dont il est question dans la première partie de ce chapitre. Ce logiciel, lui-même doté d'une interface adaptée à la population des éducateurs, lui permettra de choisir les outils et de les agencer en vue de la réalisation d'une interface particulièrement adaptée à chacune des personnes handicapées concernées.

Cependant, comme nous l'avons indiqué, la réalisation de ce logiciel à l'intention des éducateurs est un objectif trop ambitieux pour un travail d'un an. Nous ferons néanmoins oeuvre utile en créant un certain nombre d'outils biens adaptés et directement utilisables par le logiciel dont il est question ci-dessus.

Nous ne perdrons pas de vue, comme indiqué dans le dernier chapitre, que la démarche est à poursuivre pour arriver au logiciel utilisable par des éducateurs.

1.2.3.Le type d'interface choisi

Comme nous venons de l'indiquer, une collection d'outils seront à la disposition des éducateurs pour réaliser une interface adaptée aux personnes présentant un handicap. Nous définirons, dans la présente section, le type d'interface qui nous a paru le mieux adapté. Nous décrirons ainsi l'interface de type **mini-monde** et la **manipulation directe** ainsi que les avantages et les inconvénients qui les caractérisent.

A. La méthaphore du mini-monde

Selon Brenda K. Laurel [Laur, 86], quels que soient les buts de l'utilisateur, l'interface doit lui permettre de se sentir impliqué rationnellement et émotionnellement dans

le contexte de la tâche. Un peu comme on se sent impliqué dans un film ou au théâtre, à la différence près que le spectateur ne peut changer ou influencer le scénario du film ou de la pièce alors que l'utilisateur d'un système informatique est appelé à agir sur celui-ci. C'est ce que le Professeur F. Bodart appelle ***"la métaphore du mini-monde"***. Dans un tel contexte, l'utilisateur ne devra plus voir l'ordinateur comme un outil lui permettant de réaliser sa tâche, il pourra directement agir dans le mini-monde. Cependant, de la même façon que dans le monde réel nous sommes confrontés à différentes contraintes, l'utilisateur est confronté à un certain nombre de contraintes qui limitent son taux d'implication dans le mini-monde. Il existe des contraintes techniques. Par exemple, si le système ne reconnait pas la voix, il faut employer le clavier ou la souris. Et des contraintes liées à la tâche. Par exemple, nous ne pouvons réaliser deux types de tâche distincts en même temps. En général, les contraintes interdisent à l'utilisateur de sortir des frontières du mini-monde.

B. La manipulation directe

Après avoir évoqué les exigences de l'interface à offrir, il nous reste à déterminer la façon dont l'utilisateur va pouvoir agir sur celle-ci. L'utilisateur ayant un faible niveau de connaissance, il serait inopportun de le confronter à une technique d'interaction complexe. La manipulation directe est particulièrement indiquée pour ce type de population.

La manipulation directe est celle qui permet d'exécuter l'action graphiquement dans une forme proche de la manière dont on pense le problème. En manipulant des objets à l'écran, on exécute la tâche. Pas besoin de retenir des commandes et de la syntaxe, pas d'opération cachée. "What you see is what you get" (WYSIWYG) [Hutc, 86].

Selon B. Shneiderman, la manipulation directe est composée notamment :

- d'une représentation continue des objets d'intérêt,

- d'actions physiques (mouvements et sélections par souris, écran tactile,...) ou de pression sur des boutons *étiquetés* (en anglais, labelled) au lieu d'une syntaxe complexe,

- d'opérations rapides, incrémentales et réversibles pour lesquelles l'impact sur l'objet est immédiatement visible.

Le lecteur intéressé trouvera une définition de la manipulation directe dans [Shne, 87].

La manipulation directe permet donc la réduction de la distance entre le système physique et les buts de l'utilisateur d'une part, et son implication dans le système, d'autre part.

La réduction de la distance correspond à la diminution des efforts cognitifs requis par un utilisateur pour combler le gouffre d'exécution et le gouffre d'évaluation décrits précédemment. En effet, plus l'image du système représentée par l'interface est proche de la manière dont l'utilisateur pense, moins celui-ci devra faire d'effort, interpréter et développer des structures mentales pour combler les gouffres.

L'implication de l'utilisateur a déjà été examinée dans le point précédent. Au lieu de décrire des actions, l'utilisateur exécute ses actions et au lieu de recevoir une description des résultats des actions, l'utilisateur voit leur réalisation sur les objets.

Notons que, si on avait choisi un autre type d'interaction que la manipulation directe, nous n'aurions plus pu parler d'utilisation de la méthaphore du mini-monde. En effet, l'utilisateur ne pourrait agir directement sur le mini-monde. Son action sur le système serait contrariée par la nécessité d'utiliser un langage de commande réduisant à néant son implication émotionnelle et rationnelle dans le contexte de la tâche.

1. La distance

La distance qui existe entre le système physique et les buts de l'utilisateur a deux formes : sémantique et articulatoire. La ***distance sémantique*** est la relation entre la signification d'une expression dans le langage de l'interface et ce que l'utilisateur veut dire, alors que la ***distance articulatoire*** est la relation entre la signification d'une expression et sa forme [Hutc, 86].

Pour réduire la distance sémantique, le concepteur peut offrir un langage de haut niveau qui exprime directement les structures fréquemment rencontrées dans la décomposition d'un problème. Il perd alors la généralité de l'outil. Mais, s'il ne le fait pas, l'utilisateur aura des difficultés lors du choix entre les nombreuses possibilités ce qui, pour notre population, serait fatal.

Une façon de réduire la distance articulatoire, au niveau des inputs est de fournir une interface qui permette la spécification d'une action en la "mimant". Par exemple, l'action de mouvoir un curseur à l'écran peut être spécifiée en déplaçant une souris, en pointant du doigt l'écran... Au niveau des outputs, on peut réduire la distance articulatoire en représentant une action comme une modification d'objet(s). Par exemple, le changement de variables peut être mis sous forme d'un graphique plutôt que sous forme d'un tableau de

chiffres, même si, au niveau sémantique, les représentations sont équivalentes. La distance articulatoire dépend fortement de la technologie des inputs et des outputs. La souris, par exemple, est un périphérique qui permet de diminuer la distance articulatoire pour les tâches représentables spatialement.

Le langage iconique est constitué de représentations pour lesquelles la forme de l'expression est liée à sa signification. Pour cette raison, il sera abondamment utilisé dans les interfaces destinées à des personnes ayant une déficience physique et/ou mentale. Notons cependant que toute représentation est incomplète. Elle nécessitera interprétation en fonction du référentiel de l'utilisateur. Un éducateur réduira la distance sémantique pour un utilisateur particulier en mettant en oeuvre des objets familiers à cet utilisateur.

2. L'implication

Pour réussir à impliquer l'utilisateur dans un monde d'objets, il faudra que l'éducateur, intermédiaire entre le logiciel et l'utilisateur, choisisse les objets adéquats pour le domaine de la tâche. C'est-à-dire des objets réduisant au maximum la distance sémantique et articulatoire entre l'interface et l'utilisateur. Il faudra également que les opérations, lorsqu'elles sont initialisées par l'utilisateur, occasionnent des modifications dans la représentation des objets (les objets sont dès lors des entités d'input et d'output à la fois). Le choix des opérations exécutables sera effectué par l'éducateur en fonction des opérations que l'utilisateur désirera effectuer.

C. Les avantages et les inconvénients

Nous allons terminer cette section en examinant quels sont les divers avantages et inconvénients du type d'interface choisi.

Les divers avantages sont :

- Facilité de réalisation de la tâche, du passage de l'intention à l'action.

- Facilité de compréhension du système. "Ease of use". Pas besoin d'un apprentissage avant utilisation. Notons cependant que les interfaces permettant la manipulation directe ne prétendent pas prévenir les problèmes résultant d'une mauvaise compréhension du domaine de la tâche.

- Capacité de supporter la façon dont on pense au sujet d'un domaine. Cette affirmation, vue par Hutchins, Hollan et Norman comme un appauvrissement de la façon de penser (et donc comme un inconvénient), sera, dans notre situation un avantage majeur. En effet, pour une population présentant des lacunes au niveau du processeur cognitif, par l'utilisation répétée du système, nous prétendons améliorer la connaissance du domaine de la tâche.

- Certains messages d'erreurs ne sont plus nécessaires parce que les résultats sont directement visibles.

- L'utilisateur peut voir immédiatement si ses actions réalisent ses buts.

- Il semble que les utilisateurs se trouvant face à ce type d'interactions présentent moins d'anxiété parce que le système est compréhensible et parce que les actions sont réversibles. Cette caractéristique semble généralement reconnue. Elle est évidemment d'un grand intérêt pour les personnes présentant un handicap.

Les inconvénients du type d'interface choisi sont :

- Difficultés pour traiter les variables, les matérialiser à l'écran.

- Les représentations d'objets sous forme d'icônes sont incomplètes. Elles font appel au référentiel de l'utilisateur et nécessitent des connaissances ou des explications réduisant parfois la compréhension du système.

- Certains messages d'erreurs ne sont plus présentés car on suppose que la facilité d'utilisation rend évidente la non-réalisation de l'opération désirée (si on clique à l'extérieur d'un objet, rien ne se passe. On suppose donc que l'utilisateur comprendra que sa démarche est inutile ou erronée).

Chapitre 2 : Etat de l'art sommaire des divers outils d'aide à la conception d'interfaces graphiques

Introduction

Nous avons énoncé dans le chapitre précédent, les divers objectifs de notre travail. Nous allons maintenant essayer de montrer la pertinence de la voie que nous avons choisie : celle de créer une boîte à outils destinée à être exploitée par un Système de Développement d'Interfaces Usagers (SDIU) à réaliser dans le futur.

Nous débuterons par un bref historique de l'interface graphique depuis l'élaboration des concepts par le Xerox Palo Alto Research Center (Xerox PARC) jusqu'au dernières nouveautés concernant le monde du Personal Computer (P.C.). Nous examinerons ensuite un certain nombre d'éléments qui rendent l'élaboration d'une interface graphique difficile et complexe. Puis, nous introduirons quelques concepts ayant trait aux systèmes d'aide à la conception d'interfaces graphiques, que nous illustrerons par quelques exemples actuels. Nous enchaînerons par une description du travail à réaliser dans le cadre de ce mémoire ainsi que par un aperçu du futur logiciel. Nous terminerons ce dernier point en montrant que les choix que nous avons effectués sont pertinents face aux divers logiciels existants.

2.1. Bref historique de l'interface graphique.

S'il est une caractéristique à laquelle l'informatique ne peut se soustraire, c'est bien celle de la fantastique rapidité de son évolution. Voici déjà quelques lustres, lorsque l'idée de coupler des écrans aux ordinateurs prit forme, personne ne pouvait se douter des énormes transformations que cette communion allait engendrer dans le monde de l'informatique.

Dans un premier temps, les écrans ne furent utilisés que pour permettre l'affichage de résultats ainsi que la visualisation du texte introduit par le biais du clavier. Les diverses commandes que nous pouvions alors rencontrer étaient exclusivement textuelles. Elles étaient souvent propres à chaque logiciel et nécessitaient des efforts de pratique et de mémorisation intenses avant de pouvoir être maîtrisées correctement.

C'est grâce au travail de pionnier effectué au **Xerox PARC** que sont nés, dans le milieu des années 70, la plupart des concepts sous-jacents aux interfaces graphiques actuelles.

Ces concepts ont permis une phénoménale transformation dans la façon de gérer les informations. L'écran a vu ses fonctionnalités optimisées de telle façon qu'il a constitué un environnement fournissant une interaction visuellement riche pour transporter et surtout manipuler de l'information. Il n'était plus seulement un outil de visualisation, il devenait un véritable dispositif d'Entrée/Sortie (E/S). De par la matérialisation de la plupart des notions introduites par le Xerox PARC, l'écran fut doté d'objets graphiques tels que des icônes, des boutons et des barres de défilement. Le concept de ***moyens d'interaction***, défini comme "un dispositif physique à l'aide duquel un opérateur saisit de l'information ou la fait afficher" [Prov,91], concrétisé par la souris, les manettes de contrôle, les écrans tactiles, etc., a permis d'agir directement sur les objets qui étaient visualisés à l'écran. Tout ceci a suscité le concept de ***manipulation directe*** qui fut explicitement introduit un peu plus tard par Shneiderman [Shne,83].

Tous ces concepts n'ont pourtant pas connu de succès immédiat. Le Xerox STAR, fruit de l'inventivité des chercheurs du Xerox Parc, fut le premier ordinateur à matérialiser ces concepts innovateurs[Smit,82]. Malheureusement, son impact ne fut pas phénoménal.

En fait, ces idées ne se popularisèrent que, lorsqu'au milieu des années 80 et après une brève expérience avec Lisa, la firme Apple introduisit sur le marché un ordinateur qui allait entraîner de profondes mutations dans le monde de l'informatique: le Macintosh.

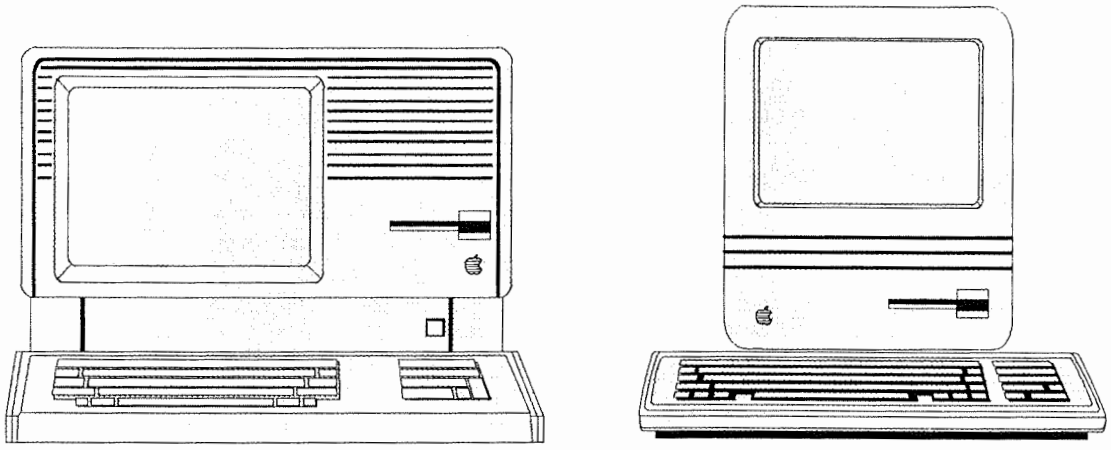


Figure 2.1. : A gauche, le Lisa. A droite, le Macintosh.

L'arrivée du Macintosh dans le monde de l'informatique eut l'effet d'un coup de pied dans une fourmilière. Grâce à l'impulsion qu'Apple fournit aux concepts du Xerox PARC, toute une série d'interfaces graphiques pour divers types de machines ont vu le jour. Citons

en quelques-unes telles Intuition pour le monde du Commodore Amiga, Graphical Environment Manager (GEM) pour le monde Atari, NextStep pour les ordinateurs Next, le Finder d'Apple, le Dosshell du MS-DOS. Toutes ces interfaces graphiques ont eu pour même objectif d'offrir aux utilisateurs une plus grande facilité dans l'apprentissage et dans l'utilisation des ordinateurs.

Pour le monde du compatible IBM, les deux interfaces graphiques les plus connues sont Presentation Manager sous système d'exploitation OS/2 et Windows sous système MS-DOS.

C'est en Novembre 1985 que fut délivrée la première version de Windows. Il allait devenir, en l'espace de quelques années, l'environnement graphique de référence pour le monde du P.C. Les versions 2.0 et 3.0 du produit, respectivement disponibles en novembre 1987 et mai 1990, eurent pour principales nouveautés la gestions de fenêtres superposées et la possibilité d'adresser de la mémoire jusqu'à 16 mégabytes.

L'émergence de la version 2.0 d'OS/2 et le divorce entre IBM et Microsoft a rendu la situation assez confuse. L'OS/2 2.0 est le premier système d'exploitation profitant pleinement de la capacité des nouvelles générations de processeurs 32 bits. La version 3.01 de Windows, sortie en mars de cette année, ainsi que ses versions précédentes sont incapables de réaliser cela. En effet, elles ont été conçues pour des machines à base de processeur 16 bits.

Pour récupérer ce retard, Microsoft promet pour fin 92, un Windows New Technology (NT) qui serait, lui aussi, un système d'exploitation 32 bits pouvant tirer un profit maximum des machines 32 bits actuelles.

2.2. Nécessité des outils

"Windows a la réputation d'être facile pour les utilisateurs mais difficile pour les programmeurs" [Petz,90].

Mais quelles sont les raisons qui rendent les interfaces graphiques si lourdes et si complexes à implémenter? Les justifications de cet état de fait sont multiples et diverses. Premièrement, il existe des raisons liées à la complexité même de l'interface graphique. Nous n'en reprendrons ici que quelques-unes:

- la **qualité du graphisme**. Ce dernier n'a fait que s'améliorer au fil des ans permettant ainsi aux concepteurs de créer des pictogrammes aisément compréhensibles et adaptables à chaque population d'utilisateurs. Un ***pictogramme***.

étant "un symbole graphique totalement statique représentant un type d'information quelconque, l'état d'un système, une option,..." [Prov,91].

Mais, lorsqu'une telle représentation est utilisée, il faut tenter d'éviter de mauvaises analogies entre pictogrammes et ne les employer que pour des objectifs précis et bien définis. Tout ceci souligne l'importance qui doit être accordée à une analyse approfondie en vue d'une représentation graphique d'actions et d'objets divers. Les pictogrammes ne représentant qu'une petite partie de l'interaction au sein d'une interface, ces remarques s'inscrivent dans le cadre de l'analyse globale de cette dernière.

- la **gestion des fenêtres** dans un environnement multi-fenêtré.

-la richesse des possibilités d'**entrée de données par le biais des moyens d'interactions,** qui sont offertes à l'utilisateur. Celui-ci peut généralement effectuer, à l'aide de la souris et/ou du clavier, un même traitement de quatre ou cinq façons différentes. Prenons l'exemple du remplissage de la boîte de dialogue reprise à la figure 2.2.. Nous pouvons répondre de diverses façons à la requête qui y est exprimée. Premièrement, nous pouvons cliquer sur un des boutons graphiques. Deuxièmement, nous pouvons sélectionner un bouton avec les touches de déplacement, puis valider notre choix en pressant sur [Return]. Le bouton sélectionné a son texte et son contour mis en évidence. Dans ce cas-ci, c'est le bouton "Oui" qui est sélectionné. Et enfin, nous pouvons utiliser les raccourcis claviers. Il suffit d'appuyer sur [Alt] et la lettre soulignée dans le libellé du bouton, pour effectuer l'action désirée.

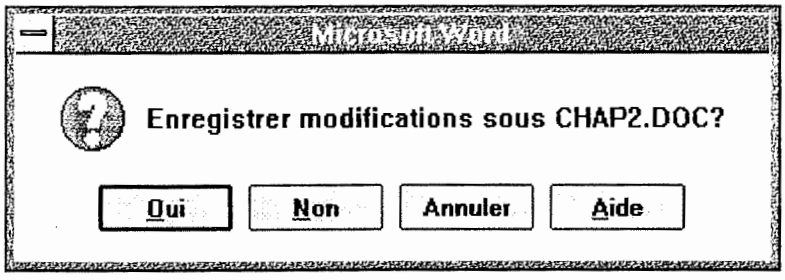


Figure 2.2. : Boîte de dialogue de confirmation d'enregistement sous Microsoft Word 2.0

Comme nous pouvons le constater, ces programmes doivent supporter une gestion concurrente des moyens d'interaction. En effet, il n'est pas possible de savoir à l'avance si une saisie se fera à l'aide du clavier ou de la souris, par exemple.

- un **feed-back sémantique** pouvant être associé à des objets. Cette notion mérite que nous nous y attardions quelque peu. Prenons un exemple propre à

l'environnement Windows. Le **Gestionnaire des Programmes** (en anglais, Program Manager) permet de gérer graphiquement des programmes lorsqu'ils sont installés sous Windows. Comme nous pouvons le voir à la figure 2.3., il est visualisé par une fenêtre qui contient elle-même plusieurs autres fenêtres représentant divers groupes d'applications tels le groupe des jeux ou le groupe de travail. Chaque groupe contient zéro, un ou plusieurs pictogrammes représentant chacun une application. Lorsqu'une de ces fenêtres est sélectionnée, sa barre de titre voit sa couleur modifiée. Le feed-back sémantique associé à la sélection d'un groupe est donc le changement de couleur de la barre de titre qui signifie que cette fenêtre est maintenant la fenêtre active.

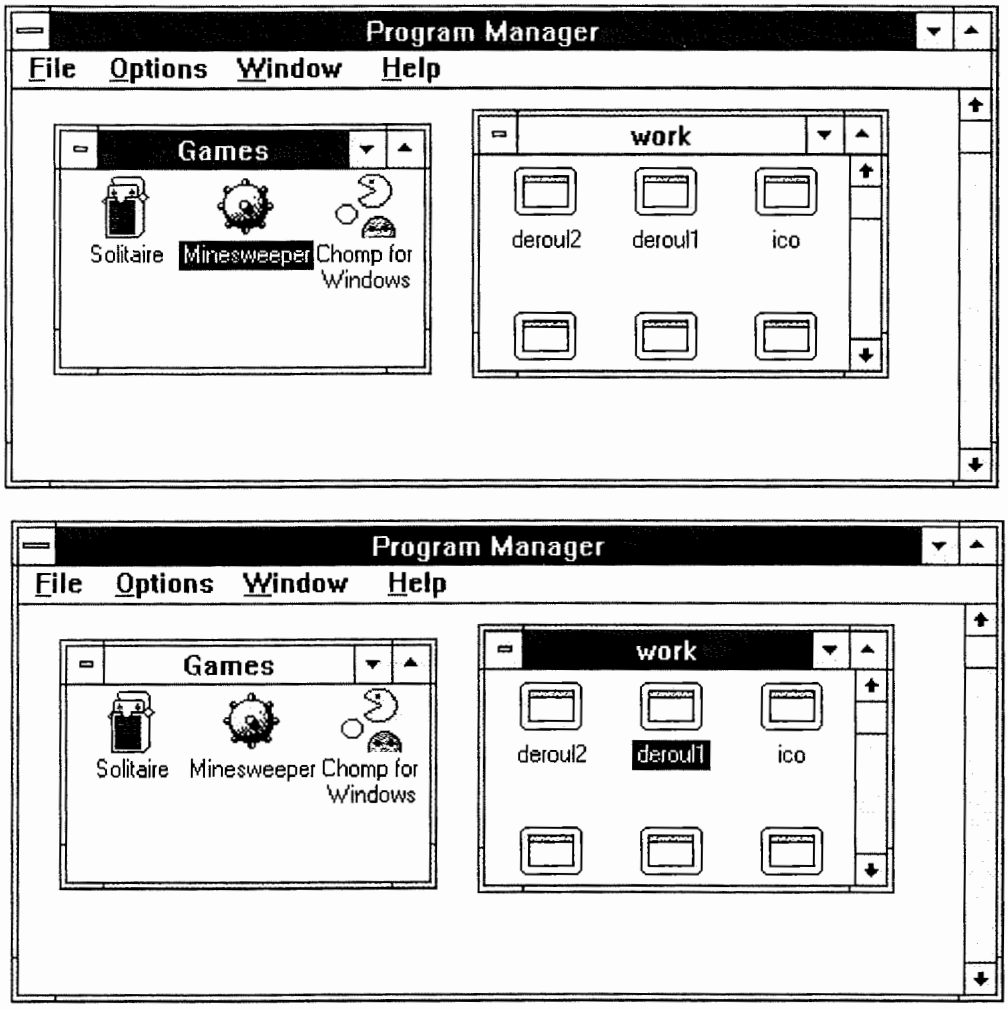


Figure 2.3. : Feed-back de la sélection de la fenêtre "WORK"

-La **programmation événementielle** des interfaces graphiques. En effet, ce type d'interface nécessite une programmation événementielle qui oblige le programmeur à concevoir ses applications de manière différente. La structure de ces

programmes ne correspond plus à la structure séquentielle, plus couramment répandue.

Pour clôturer ce premier point, nous pouvons affirmer que ce n'est pas un programmeur Windows qui se doit de savoir gérer plus de 900 définitions et macros, 550 fonctions, 230 messages différents et plus de 60 types de données qui infirmera le fait que ce type d'interface soit assez lourd à élaborer.

Une seconde explication quant à la lourdeur des interfaces au niveau de leur implémentation est le lien étroit qui existe entre la population d'utilisateurs et l'interface.

Si nous trouvons assez bien de littérature sur les lignes de conduite à adopter pour élaborer correctement un certain type d'interfaces, nous ne retrouvons pratiquement pas d'outils permettant d'évaluer correctement l'adéquation d'une interface à une population. Benard Senach [Sena,90], nous propose toute une série de travaux de recherche tentant d'évaluer de façon systématique la qualité ergonomique d'une interface. Il conclut lui-même qu'aucun des travaux présentés ne répond de manière satisfaisante à cette recherche et n'est en mesure d'établir un diagnostic complet d'une interface utilisateur.

La seule technique qui semble aujourd'hui la plus fiable est celle de la ***conception par raffinements successifs*** (en anglais, iterative design). Cette méthode consiste à implémenter différents prototypes d'interfaces pour une même application et à les tester auprès des utilisateurs. En fonction du retour d'information obtenu lors de ces tests, les prototypes seront améliorés jusqu'à obtention des résultats désirés. Il est fort possible qu'arrivés à ce stade, les concepteurs constatent l'impossibilité d'élaborer une interface unique adaptée aux utilisateurs car ces derniers constituent un ensemble de populations fort diversifié. Nous pouvons dès lors assister à un double scénario :

- soit il est possible de trouver une interface qui corresponde de manière optimale aux exigences de l'ensemble des utilisateurs. Nous assistons alors à la production d'un logiciel unique avec une interface bien ciblée.

- soit cela n'est pas possible. Nous assistons alors à la production de plusieurs prototypes d'interface, chacun d'entre eux donnant naissance à une variante de l'application soigneusement adaptée à la population à laquelle elle s'adresse.

Or, les 40 à 50 % du code d'une application sont exclusivement réservés à des aspects d'interfaces [Bobr,86]. Donc, aussi logique que puisse sembler cette méthode de conception, elle n'en demeure pas moins extrêmement coûteuse et peu de firmes peuvent se permettre d'investir de gros budgets dans l'élaboration d'une interface optimale. Pour répondre néanmoins à l'attente des utilisateurs, nous assistons à l'émergence d'une "nouvelle génération d'applications telles que Microsoft Excel, Samna Ami Professional et Microsoft Word qui permettent de réaliser des raccourcis personnalisés, d'altérer des menus et de changer certaines caractéristiques par défaut. Ces applications sont conçues pour des environnements complexes à interfaces graphiques et offrent un remarquable degré d'adaptabilité. Certains développeurs et consultants pourront donc satisfaire les diverses exigences des utilisateurs finaux simplement en étendant et en adaptant des logiciels plutôt qu'en développant du software pour chaque demande" [Eisn,90].

A titre d'exemple, citons le cas du Microsoft Word 2.0. Ce logiciel possède toute une série d'options qui permettent une adaptation du logiciel à l'utilisateur. Word permet notamment de redéfinir les boutons et les fonctionnalités de la barre d'outils, de modifier les menus, d'effectuer des sauvegardes automatiques après un certain intervalle de temps, de modifier la présentation du texte à l'écran, de changer des raccourcis, etc..

Ce n'est donc pas une mince tâche que de réaliser une interface graphique remplissant correctement les objectifs qui lui ont été assignés. Au vu de cette problématique, le besoin s'est rapidement fait sentir d'élaborer des programmes d'aide à la conception de telles interfaces. Des outils tels des éditeurs de dialogue, des éditeurs de ressources, des générateurs de code pour interfaces et des bibliothèques de procédures sont apparus. Comme le nombre de ces applications est en perpétuelle croissance, il est parfois très ardu de s'y retrouver. Nous proposons dans la section suivante un éclaircissement en la matière en effectuant un modeste état de l'art des outils d'aide à la conception d'interfaces sous Windows, selon une classification propre à Brad Myers [Myer,89].

2.3. Classification de divers logiciels d'aide à la réalisation d'interface graphiques

Nous ne reprendrons, dans cette section, que des outils propres à l'environnement Windows. Les logiciels suivants ont pu être visionnés en version de démonstration. Il s'agit

de Visual Basic de Microsoft Co., Toolbook d'Asymétrix, ProtoView de ProtoView Development Co., le WhiteWater Ressource Toolkit (WRT) du WhiteWater Group et eXtended Virtual Toolkit (XVT) de l'Advanced Programming Institute. Nous avons repris ces divers logiciels car ils illustrent les divers concepts qui seront introduits par la suite. De plus, des références à ces applications ont été fréquemment trouvées dans la littérature courante.

Notez que, dans la section suivante, lorsque nous parlerons d'outils, ce terme sera compris dans son sens le plus général c'est-à-dire un moyen servant à effectuer un certain travail. Cela pourrait tout aussi bien être une méthode, un logiciel, une primitive, une théorie,... Par contre, lorsque nous traiterons de boîte à outils, nous entendrons par là un ensemble de routines permettant d'effectuer un certain traitement.

2.3.1. Les catégories d'outils

Myers considère que les outils d'aide à la conception d'interfaces usagers peuvent être regroupés en deux grandes familles : les **boîte à outils** (en anglais, toolkit) et les **Systèmes de Développement d'Interfaces Usagers** (SDIU, en anglais UIDS pour User-Interface Development System).

"Une ***boîte à outils*** pour interfaces usager est une bibliothèque de techniques d'interaction. Une technique d'interaction étant une manière d'utiliser un moyen d'interaction d'entrée afin d'introduire une valeur telle qu'une commande, un nombre ou un nom, qui engendre un retour d'information (en anglais, feed-back) à l'écran" [Myer,89].

La boîte à outils sert donc principalement de base aux programmeurs pour pouvoir contrôler les diverses interactions que l'utilisateur peut effectuer par le biais des périphériques d'entrées. Ces techniques d'interaction constituent donc le regroupement d'un ensemble d'objets interactifs (boutons, boîtes de dialogue, champs d'édition,...) et des moyens d'interactions afin de permettre à un utilisateur d'effectuer des saisies d'informations et d'examiner le retour d'information conséquent.

Par exemple, plutôt que d'introduire textuellement une commande, une primitive d'une boîte à outils permettra de créer un bouton graphique qui, lorsqu'il sera cliqué par la souris, effectuera la commande.

"Le ***SDIU***, quant à lui, est un ensemble intégré d'outils qui aide le programmeur dans la conception et la programmation de la plupart des aspects de l'interface" [Myer,89]. Le SDIU constitue donc un support non seulement pour la conception mais également pour l'implémentation. C'est notamment ce côté implémentation qui le distingue du ***Système de Gestion d'Interface Usager*** (SGIU, en anglais, UIMS pour User-Interface Management System), ce dernier étant généralement considéré comme "un outil ou un ensemble d'outils conçu pour faciliter le façonnage et la gestion de la conversation au travers des moyens d'interaction, des techniques d'interaction et des styles d'interaction". Un ***style d'interaction*** étant "une combinaison logique de techniques d'interaction en vue d'organiser de manière unifiée et cohérente une suite finie de saisies/affichage" [Prov,91].

2.3.2 Les boîtes à outils

Il y a deux catégories de boîtes à outils.

La première, la plus courante, est "constituée d'**une collection de routines** qui peuvent être **appelées par un programme**". A titre d'exemple, nous pouvons citer **eXtensible Virtual Toolkit** (XVT) de l'Advanced Programming Institute. Cette boîte à outils a ceci de particulier qu'elle est constituée d'un ensemble de routines qui permettent à leurs utilisateurs d'implémenter des logiciels qui pourront être portés sur divers systèmes et/ou machines. Actuellement, XVT permet la portabilité de logiciels entre compatibles IBM fonctionnant sous Windows et Macintosh. Elle est constituée d'une bibliothèque uniformisée de procédures écrites en langage C et peut être vue comme une fine couche entre le système sur lequel reposent ses procédures et l'application. Les diverses fonctionnalités qu'elle offre sont classiques des environnements graphiques. La création et la manipulation de fenêtres, la création d'objets graphiques à l'écran, l'allocation de blocs de mémoire, la gestion de boîtes de dialogue, la gestion de fichiers et de répertoires ainsi que la gestion du clipboard constituent la plupart des fonctionnalités offertes. Il suffit, pour travailler avec ces dernières, d'utiliser les fichiers d'en-tête (en anglais, header) de XVT en remplacement des fichiers de base et d'employer en plus des bibliothèques de base, la bibliothèque XVT. Cette dernière est bien entendu différente pour chaque environnement.

Le second type de boîte à outils "**utilise un style de programmation orientée objet** avec héritage, ce qui rend plus aisée l'adaptation des techniques d'interaction" [Myer,89]. L'exemple type de ce genre de boîte à outils, ce sont les trois **bibliothèques à liens dynamiques** (BLD, en anglais DLL pour Dynamic Link Library) qui contiennent les fonctions Windows. Ces BLD se nomment Kernel, User et GDI et leurs fonctionnalités sont reprises à la figure 2.4.

Nom du module	***Fonctions supportées***
GDI.EXE	Interface graphique : sortie des images graphiques, gestion de la palette de couleurs
USER.EXE	Fenêtre, icône et gestion du curseur
KERNEL.EXE	Gestion de la mémoire, de la liste de tâches

Figure 2.4. : les trois principales Bibliothèques à Liens Dynamiques de Windows

Ces bibliothèques permettent effectivement un mode de programmation proche de l'orienté objet. Windows offre, pour toute une série de techniques d'interaction, un comportement générique. Au moyen des diverses fonctions qui sont offertes au programmeur, il est loisible de transformer à souhait ces comportements et/ou d'hériter du comportement générique de Windows pour les cas désirés.

Notons encore que ces bibliothèques ont un sens particulier. Elles sont différentes des bibliothèques "classiques" qui contiennent le code objet de modules qui sera recopié, lors du chaînage (en anglais, linkage), en lieu et place de l'appel qui a été réalisé dans l'application. Les ***BLD*** sont le fruit d'une nouvelle forme de chaînage effectué lors de l'exécution du programme. Elles sont en fait une "nouvelle forme d'objets exécutables qui ressemblent fortement aux programmes chaînés mais possèdent en plus des informations sur certains points d'entrée permettant ainsi l'accès à leurs fonctions. Une BLD est donc utilisée comme une bibliothèque conventionnelle excepté que, lorsque l'on travaille avec elle, le linker ajoute une référence vers un point d'entrée de la bibliothèque au lieu d'ajouter du code et des données déjà compilés" [Eisn,90]. Ce n'est que lors de l'exécution du programme que ces bibliothèques seront chargées en mémoire centrale et que les fonctions désirées seront exécutées. Bien que ces BLD soient stockées sous la forme de fichier .EXE, elles ne peuvent s'exécuter d'elles-mêmes.

Les boîte à outils, bien que facilitant la programmation des interfaces graphiques, possèdent cependant un désavantage majeur. Il est en effet courant de constater que, pour pouvoir faire appel à une procédure en provenance de bibliothèques, il faut parfois une quantité de code préalable impressionnante. La facilité d'emploi de ces routines de même que leur élaboration ne semble donc pas toujours évidente. Nous pouvons néanmoins ajouter que ce malaise se réduit fortement du fait d'une certaine standardisation.

2.3.3 Les Systèmes de Développement d'Interfaces Usagers

Les problèmes rencontrés par les programmeurs lorsqu'ils utilisaient des boîte à outils ont été à l'origine de l'élaboration des SDIU. Citons le cas d'Apple, qui créa le SDIU MacApp après avoir constaté la difficulté des utilisateurs à employer la boîte à outils du Macintosh.

Comme nous l'avons vu à la section 2.3.1, un SDIU doit supporter la manipulation de tous les aspects d'une interface, afin de permettre au concepteur de combiner et de séquencer les techniques d'interaction. Pour ce faire, un SDIU peut être composé de divers éléments:

-"une **boîte à outils**

- une **composante chargée du contrôle du dialogue** qui permette de gérer les séquences d'événements et les techniques d'interaction

- une **charpente de programmation** qui guide et structure le code de l'interface et la sémantique de l'application

- un **éditeur** permettant la manipulation par souris

- une **composante d'analyse** qui puisse soit évaluer l'interface sur base de règles ergonomiques ou de lignes de conduite ou sauver des informations sur l'interaction avec l'utilisateur qui seront sujettes à évaluation" [Myer,89].

En fonction de la complexité et des objectifs assignés à chaque SDIU, ce dernier comprendra tout ou seulement quelques-uns des éléments constitutifs susmentionnés.

La classification des SDIU peut être effectuée selon divers critères. Nous avons choisi de classifier chaque système selon la façon dont la spécification de l'interface peut être établie. Nous retrouvons donc des SDIU :

- autorisant une spécification basée sur des langages,

- générant automatiquement une interface en se basant sur la spécification de procédures ,

- permettant une spécification graphique.

Nous ne dirons que quelques mots des deux premières classes, la troisième constituant notre principal centre d'intérêt.

A. Les SDIU basés sur un langage

Cette classe d'applications, comme son nom l'indique, est **supportée par un type de langage dédicacé**. Par dédicacé, nous entendons que la sémantique et la syntaxe de ce

dernier sont basées sur les objectifs qui lui ont été assignés. Ces langages peuvent prendre différentes formes. Diagrammes état-transition, langages déclaratifs, langages orientés événements et langages orientés objet constituent globalement les divers types de langages que nous pouvons retrouver dans la littérature courante. Les programmeurs utilisent donc ces divers langages pour spécifier et élaborer une interface. Pour illustrer par quelques exemples cette catégorie de SDIU, nous pouvons citer Rapid/USE pour les diagrammes état-transition, Open Dialogue pour les langages déclaratifs, Algae pour les langages orientés événements et MacApp pour les langages orientés-objet[1].

B. Les SDIU à génération automatique

Cette seconde catégorie constitue encore un domaine en pleine expérimentation. Ces SDIU permettent d'élaborer une interface **à partir de spécifications de procédures**. Ce type de système analyse donc chaque routine et pour chaque définition génère une interface. La difficulté, résidant dans l'utilisation d'une boîte à outils ou d'un SDIU pour concevoir et/ou implémenter des interfaces, se trouverait de ce fait pratiquement réduite à néant. Il suffirait, lorsque l'interface résultante ne rencontre pas les desiderata de l'utilisateur, de la modifier à l'aide du système qui a servi à la concevoir. Citons Interactive Transaction System (ITS), Judgement-based Automatic Dialog Editor (JADE) et User Interface Development Environment (UIDE) pour exemples[2].

C. Les SDIU à spécification graphique

Cette dernière famille de SDIU, basée sur la spécification graphique, est probablement la plus appropriée à l'élaboration d'interfaces usagers. Les outils appartenant à cette catégorie permettent **la spécification et le placement des objets constitutifs de l'interface par un éditeur de présentation à manipulation directe**. Mentionnons PERIDOT par exemple, qui est un outil élaboré par Myers.

Etant donné le caractère visuel d'une interface, ce type de système semble tout à fait adapté et permet, en plus, une meilleure compréhension et utilisation par un concepteur.

a. le WhiteWater Resource Toolkit et ProtoView.

Un **premier sous-type de SDIU** à spécification graphique, est constitué par l'ensemble des systèmes qui "permettent le placement à l'écran des techniques d'interaction

1 Les références d'articles traitant de ces systèmes pourront être trouvées dans les notes bibliographiques.
2 idem note1

telles que menus, barres de défilement, boutons, etc. . Ils laissent également la possibilité de spécifier des zones supplémentaires pour du graphisme d'E/S spécifique à l'application" [Myer,89]. Après l'élaboration de la partie visuelle de l'interface, il faudra encore créer toutes les routines qui seront exécutées lors de chaque action sur les objets interactifs de l'interface.

Dans cette catégorie de SDIU, nous retrouvons le **WhiteWater Resource Toolkit** (WRT) et **ProtoView**. ProtoView étant un outil plus complet et plus complexe que le WRT, nous commencerons notre description par le WRT.

Notez ici que l'emploi du mot "Toolkit" dans le WhiteWater Resource Toolkit est malheureux. En effet, en aucun cas le WRT ne correspond à la boîte à outils définie par Myers.

Le WRT est un éditeur de ressources. "Les ***ressources*** sont des objets de données qui sont assemblés au fichier .EXE d'une application et sont utilisées par cette dernière au besoin" [Adle,91]. Ces objets peuvent être des boîtes de dialogue, des menus, du texte, des bitmaps ou encore des icônes ou des curseurs.

Nous pouvons par ailleurs examiner à la figure 2.5., la composante de spécification d'une boîte de dialogue avec les divers éléments constitutifs de son interface et une boîte de dialogue créée dans sa zone de travail.

En élaborant des SDIU tel le WRT, les principales finalités des concepteurs étaient, d'une part, de réduire le laps de temps qu'il faut à un concepteur pour constituer les éléments graphiques de son interface et, d'autre part, de tirer parti des avantages que peut fournir le concept de ressources.

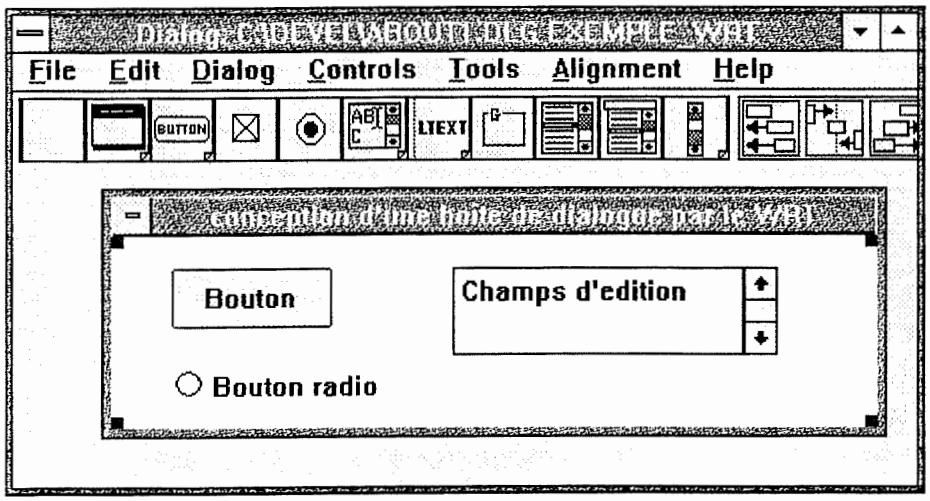


Figure 2.5. : l'éditeur de boîtes de dialogue.

Les principaux intérêts que les ressources offrent sont :

- leur localisation sur disque et leur chargement uniquement sur demande, ce qui permet de diminuer parfois considérablement la place mémoire que nécessite une application

- l'inutilité de recompiler le code source d'une application si une modification de la ressource associée a lieu. Les fichiers de ressource étant intégrés au fichier exécutable par des compilateurs de ressources, il suffit dès lors de recompiler la ressource et de la rattacher au fichier .EXE.

Tout ceci se traduit par une plus grande modifiabilité des objets interactifs d'une interface homme-machine au niveau de la présentation, de la langue et du placement à l'écran. Les principales composantes et fonctionnalités du WRT sont les suivantes :

- un **éditeur de boîtes de dialogue** qui permet à l'aide de la souris de placer dans une boîte de dialogue les différents contrôles que Windows offre et de modifier certaines de leurs propriétés. Les ***contrôles*** peuvent être définis comme des objets interactifs Windows tels les boutons, les champs d'édition, les zones d'affichage, les listes de sélection, etc..,

- un **éditeur de curseurs, d'icônes et de dessin** qui ressemble à un logiciel de dessin simplifié,

- un **éditeur de menus** qui permet de définir une barre de menu avec ses menus déroulants et les raccourcis pouvant être associés à chaque item ainsi que d'autres caractéristiques,

- un **éditeur de texte,**

- un **éditeur de raccourcis**. Ceux-ci permettent lorsque l'utilisation de la souris devient fastidieuse pour certaines commandes, d'utiliser le clavier avec des combinaisons de touches simples et rapides.

Lorsque les ressources auront été éditées, il faudra encore créer le code qui permettra de les afficher et, pour chaque technique d'interaction, développer des routines qui puissent les gérer. Les références aux ressources seront, quant à elles, réalisées par le biais d'identifiants qui leur seront assignés lors de leur édition.

Nous allons maintenant aborder la description de ProtoView. Il se compose de quatre parties:

- **ViewPaint** qui permet, comme le WRT, d'éditer une fenêtre ou une boîte de dialogue en positionnant de manière interactive des contrôles. Ce que ViewPaint possède de plus, c'est un générateur de code qui autorise la création des divers fichiers source qui faciliteront la création des routines permettant de gérer les fenêtres. En effet, chacune des spécifications graphiques d'une fenêtre donnera lieu à une charpente de code qui permettra au développeur d'y associer des routines de gestion. Nous pouvons, à la figure 2.6., examiner un exemple de fichier source élaboré par ViewPaint,

```
    ABOUT1 DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE
67, 54, 150, 49
    STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU | 0x80L
    CAPTION "Boîte de dialogue éditée par ProtoView"
    BEGIN
    /* MainWndProc entrywun,@M@V,  */
      CONTROL   "Proto"   101,   "BUTTON",   WS_CHILD   |
WS_VISIBLE | WS_TABSTOP, 9, 12, 73, 14
      CONTROL   "View"   102,   "BUTTON",   WS_CHILD   |
WS_VISIBLE | WS_TABSTOP, 22, 34, 86, 12
    END
    /* Color values and validation strings */
    entrywindow RCDATA LOADONCALL MOVEABLE DISCARDABLE
    BEGIN
    1,"@BITMAP(c:\windows\arcade)",0,
    ...
    END
```

Figure 2.6. : fichier source d'une boîte de dialogue élaborée par ProtoView.

- **ProtoGen** qui permet de définir la fenêtre principale de l'application Windows, de créer des menus et d'associer toutes les composantes de l'interface élaborées à partir de ViewPaint, de tester et de modifier ces dernières s'il y a lieu. Lorsque tout ceci est réalisé, ProtoGen génère le fichier source en C ou C++ ainsi que tous les fichiers nécessaires au compilateur pour élaborer correctement une application Windows,

- une BLD appelée **ProtoView** qui comprend 175 fonctions,

- une BLD nommée **Control** qui contient 12 nouveaux types d'objets de contrôle ainsi que les fonctions associées.

Comme nous pouvons le constater, ProtoView va beaucoup plus loin que le WRT. S'il offre tout comme ce dernier un placement des contrôles de manière interactive, il permet de générer du code qui servira de base au codage final de l'application. De plus, ProtoView offre quelques objets de contrôle supplémentaires ainsi que 175 fonctions qui, selon les dires de ses concepteurs, sont suffisantes pour élaborer la plupart des applications graphiques.

b. Toolbook et Visual Basic

La seconde sous-catégorie de SDIU à spécification graphique est constituée de l'ensemble des "**systèmes organisant une interface comme un réseau de pages statiques**. Chaque page contient du texte, des graphismes et des techniques d'interaction qui permettent au système d'effacer des pages ou de se diriger vers d'autres pages. Généralement, ces systèmes requièrent le codage des techniques d'interactions dans un langage de programmation conventionnel" [Myer,89].

Toolbook, logiciel hypermédia, appartient à cette catégorie. Visual Basic, bien qu'allant plus loin que la définition de Myers, fera également partie de cette catégorie . Examinons en premier lieu Toolbook.

"Toute ressemblance avec un célèbre logiciel pour Macintosh... n'a en fait rien d'une coïncidence : Toolbook est exactement l'hypercard de Windows" [Cass,90].

S'il semble évident que les concepteurs de Toolbook ont tenu à préserver intactes les principales caractéristiques qui ont fait le succès d'Hypercard, il faut néanmoins souligner que ce dernier est plus évolué que Toolbook. Mais, la ressemblance est telle, que des logiciels comme ConvertIt permettent une conversion automatique des "**piles**" d'Hypercard en "**livres**" Toolbook.

Toolbook est donc un logiciel hypermédia. Tout comme le concept d'***hypertexte*** permettait "d'associer à un mot d'un texte, un autre mot ou un autre texte par un lien, l'***hypermédia*** offrira la même fonction mais pour des informations qui peuvent être textuelles, graphiques ou encore sonores" [Cass,90]. En réalisant des actions, tel le click de la souris sur certains objets, nous voyons apparaître un texte, un graphique ou nous entendons un son numérisé se rapportant à l'objet.

Là où Hypercard gère des "**cartes**", Toolbook fait de même mais avec des "**pages**". Un ensemble de "pages" reliées entre elles constitue, bien entendu, un "livre". Dans chaque page, nous pouvons positionner du texte, des boutons ou encore des dessins. Comme on peut le voir à la figure 2.7., chaque page est constituée de deux niveaux. Le premier comprend la forme de la page. Par exemple, pour la réalisation d'un annuaire téléphonique, la forme comprendra les champs nom, adresse et numéro de téléphone ainsi que les quelques boutons nécessaires au défilement des pages.

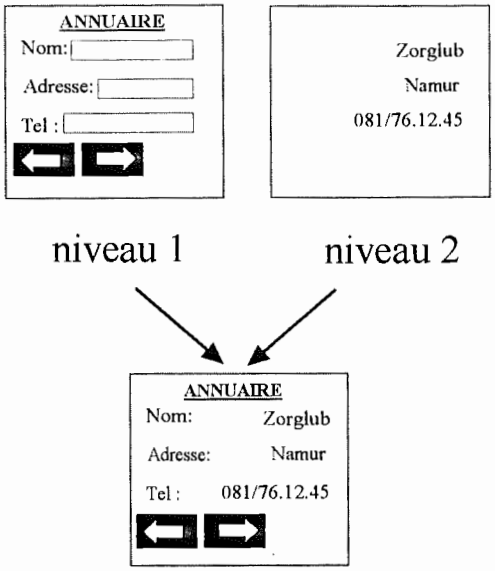


Figure 2.7. : Description des niveaux d'une "page" Toolbook

Chaque nouvelle page contiendra donc d'office tous les champs formatés. Au second niveau, nous ne retrouverons que des éléments propres à la seule page, le contenu

A chaque objet sera associé un script décrivant le traitement à effectuer en réaction à un événement donné. Ce script sera réalisé en OpenScript qui, tout comme Hypertalk, est un langage propre à l'hypermédia auquel il se réfère. Nous noterons ici quelques fonctionnalités supplémentaires propres à Toolbook. Celui-ci permet de définir un script pour un groupe d'objets afin que ces derniers puissent réagir de la même manière lors de déclenchements d'événements identiques. Plus intéressante encore est la fonctionnalité "Enregistrement" qui permet de traduire directement les actions sur des objets, en texte OpenScript. Cela permet de développer du code de manière interactive. Il ne reste plus qu'à l'associer aux objets auxquels il se rapporte. Toolbook peut également étendre ses fonctionnalités par adjonction de routines sous formes de bibliothèques à liens dynamiques. Nous pouvons encore signaler l'existence du module "Author Resource Kit", qui permet de lire un "livre" sans qu'il soit nécessaire d'utiliser Toolbook, qui prend énormément de place disque et de place mémoire.

Notez pour terminer ce point que, tout comme Hypercard, l'objectif de Toolbook n'est pas à proprement parler de créer des applications mais bien de servir de langage commun à tous les possesseurs de P.C. sous Windows pour diffuser des informations, créer des didacticiels ou présenter de nouveaux produits.

Examinons maintenant **Visual Basic** (VB).

VB est un langage de programmation qui "combine l'interface usager de Windows avec une version du langage de programmation Basic" [Eule,91]. Bien que VB n'en soit

qu'à sa première version, il constitue un progrès considérable en matière de programmation. Il n'est en effet plus question de programmer des lignes et des lignes de code. VB permet une programmation interactive basée sur le modèle orienté événements de Windows. Nous pouvons examiner à la figure 2.8., l'interface de VB.

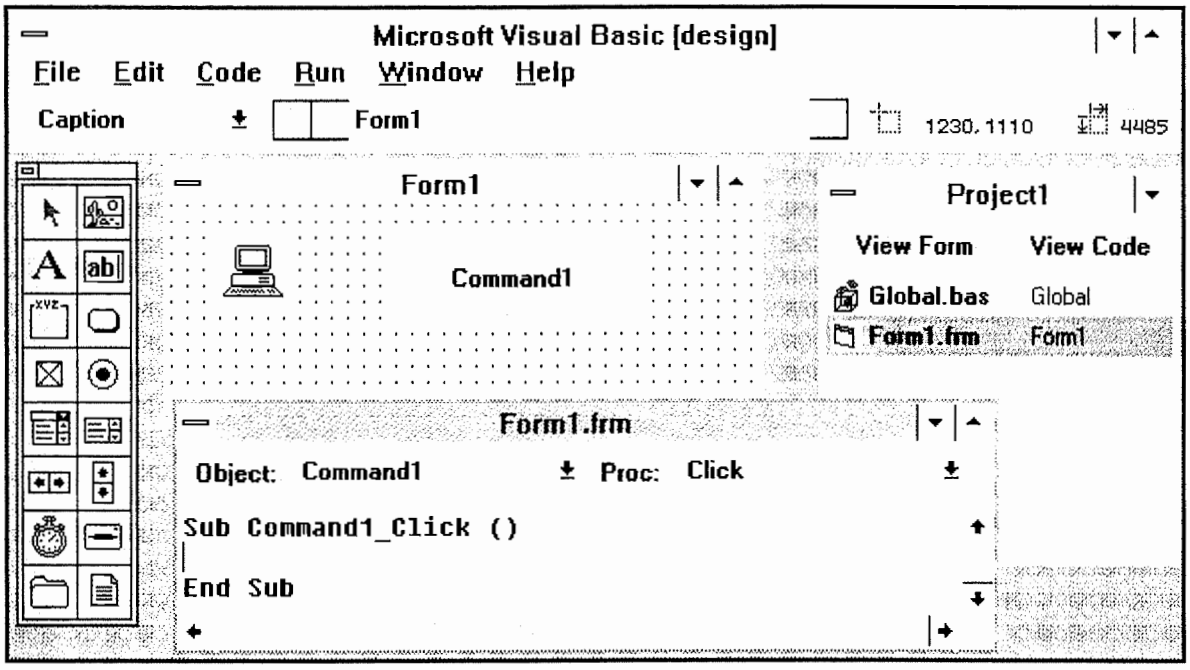


Figure 2.8. : L'interface graphique de Visual Basic.

Il existe deux grandes classes d'objets graphiques en VB : les formes et les contrôles. Les ***formes*** correspondent aux fenêtres et aux boîtes de dialogue tandis que les ***contrôles*** sont les objets graphiques classiques tels les boutons, les champs d'édition, etc., plus quelques autres apportés par VB. Une application en VB est basée sur un ensemble de formes dans lesquelles nous pouvons positionner divers contrôles. Chaque propriété d'un type d'objets peut être définie très aisément de manière interactive. L'ensemble des propriétés est bien plus complet que ce que nous avons pu retrouver dans d'autres outils. La forme "Form1" à la figure 2.8. représente l'interface d'une application en cours d'élaboration. Nous pouvons voir dans cette fenêtre le bouton "Command1" et une image.

A chaque type d'objets est également associé un ensemble d'événements qui lui correspond, chacun d'entre eux étant lié à une action de l'utilisateur ou du système. S'il peut être regrettable que VB ne permette pas l'accès à tous les événements possibles pour chaque type d'objets, le sous-ensemble qu'il supporte reste néanmoins très développé permettant de réaliser des applications très poussées. Dans le cas de la figure 2.8., nous pouvons examiner qu'à la fenêtre Form1.frm est demandé le script d'une action à effectuer lorsque le bouton "Command1" est cliqué.

De plus, VB permet, grâce à un "mode interpréteur", d'exécuter à tout moment les traitements qui ont été associés à chaque événement permettant ainsi le développement interactif et itératif d'une application. Lorsque les traitements ont été vérifiés, le programmeur peut obtenir une version exécutable de son programme. Bien que les liens entre les langages C et Pascal et le langage Basic soient délicats à concevoir, VB permet d'étendre ses fonctionnalités par l'utilisation de Bibliothèques à Liens Dynamiques (BLD). Malheureusement, il ne peut ni créer des BLD, ni fournir du code qui puisse faire l'objet d'une bibliothèque en code source. Ceci tient au fait qu'il n'y a pas à proprement parler de code source en VB. Il n'est pas possible de visionner le code d'un programme comme nous pourrions le faire avec d'autre logiciels de programmation. VB maintient en plus du code introduit pour chaque événement, toute une série de variables internes ayant rapport avec les objets élaborés, qui lui serviront à élaborer la version exécutable du programme.

2.3.4 Conclusion

Comme nous avons pu le constater, les logiciels d'aide à la conception des interfaces graphiques sont dorénavant plus qu'une réalité. Ils sont diversifiés, relativement puissants et permettent, dans la plupart des cas, d'aborder la programmation d'une manière plus aisée.

Pour réaliser les outils que nous verrons plus en détail dans les chapitres suivants, nous avons été amenés à effectuer certains choix. Nous proposons dans la section qui suit de nous attarder un peu sur les raisons de ces choix.

2.4. Les choix effectués et leur pertinence

2.4.1. Les outils graphiques et le logiciel

Les outils que nous allons réaliser cette année constitueront une boîte à outils qui permettra d'aider un programmeur à réaliser une interface pour personnes handicapées.

Ces outils sont en fait une série de primitives permettant de gérer un ensemble d'objets interactifs. Ces objets, nous les avons repris des divers mémoires pour personnes handicapées réalisés précédemment. Nous en faisons une spécification Orientée Objet (O.O.) dans le chapitre 4. Les primitives, elles, permettront à un informaticien de ne pas devoir reprogrammer certaines parties de son interface chaque fois qu'il doit réaliser une nouvelle application.

Notez bien que cette boîte à outils correspondra en tous points à la définition de Myers c'est à dire "une collection de routines pouvant être appelées par un programme".

Quant au futur logiciel, ce sera un programme permettant d'élaborer de petits exercices de manière interactive. Cette application constituera un SDIU à spécification graphique. Il sera constitué de divers composants :

- notre **boîte à outil** qui supportera la création des objets graphiques nécessaires plus d'autres outils réalisés ultérieurement,

- un **éditeur de dialogue** pour permettre la spécification des exercices par la souris,

- une **composante de paramétrage** qui permettra l'adaptation des exercices au divers moyens d'interactions utilisés par les éducateurs.

- une **composante de contrôle de dialogue**, l'interpréteur permettant d'exécuter le dialogue issu des interactions entre les exercices et les personnes handicapées,

- et vraisemblablement une **composante de contrôles syntaxiques et sémantiques** qui permettra de vérifier si la syntaxe et la sémantique des exercices sont correctes.

Une spécification détaillée quant à notre vue du logiciel sera donnée dans le chapitre 7.

2.4.2 Pertinence de la voie choisie

Pour réaliser la boîte à outils dont nous parlons à la section précédente, nous avons dû opérer certains choix.

Ces choix sont-ils pertinents? N'existe-t-il pas des bibliothèques contenant la plupart des objets graphiques que nous avons réalisés? Un logiciel d'élaboration d'exercices serait-il vraiment utile alors que tant de nouveaux outils pour gérer des interfaces graphiques ne cessent de se développer? Nous allons tenter d'apporter des réponses à ces différentes questions.

Le choix du langage peut ne pas sembler pertinent. En Octobre 91, a été mise en vente une version de **Turbo Pascal** permettant la programmation d'application Windows. Cette version autorisait même une programmation simplifiée du traitement des ressources et des contrôles grâce à l'utilisation de fonctions appelées **Application Programming Interface** (API) plus générales que les fonctions Windows. Malheureusement, cette API ne

nous était d'aucune utilité. En effet, nos objets nécessitaient une programmation de bas niveau et les traitements qui auraient pu être simplifiés, grâce à ces fonctions, auraient été négligeables. Il faut également considérer le fait que toute la documentation que nous avions à notre disposition traitait de la programmation Windows en C. Il aurait donc fallu que nous effectuions, pour tous nos documents, une transition du C vers le Pascal. L'apprentissage de Windows nous ayant déjà pris quelques mois, il ne nous a pas été possible d'effectuer ce travail supplémentaire.

Visual basic semblait, lui aussi, être un bon outil pour nous permettre d'élaborer nos objets. Malheureusement, comme nous l'avons précisé à la section 2.3.3., il ne permet ni l'élaboration de bibliothèques, ni de disposer du code source d'une application. Il ne nous est donc d'aucune utilité dans notre travail.

Nous pouvons ajouter à propos de nos objets, que bien que nous n'ayons pu visionner toutes les bibliothèques Windows existantes, nous doutons fortement d'y retrouver nos objets tant ils sont spécifiques au problème qui nous est posé.

La création d'un Système de Développement d'Interfaces Usagers qui exploitera notre boîte à outils nous semble également pleinement justifiée.

Primo, il constitue la réponse à une véritable demande. Notre stage en Grande-Bretagne et notre visite au "Trèfle" n'ont fait que nous conforter dans cette idée.

Secundo, il n'existe pas, à notre connaissance, de logiciel qui permette l'élaboration de petits exercices adaptables à une interaction avec des personnes handicapées, sans devoir faire appel à un quelconque langage de programmation. Le logiciel en question serait dépourvu de cette notion et serait basé sur une **élaboration interactive**. Pour illustrer la signification d'"adaptables à une interaction avec des personnes handicapées", prenons comme exemple l'exercice montré à la figure 2.9. . Dans cet exercice, la personne handicapée doit rechercher un intrus.

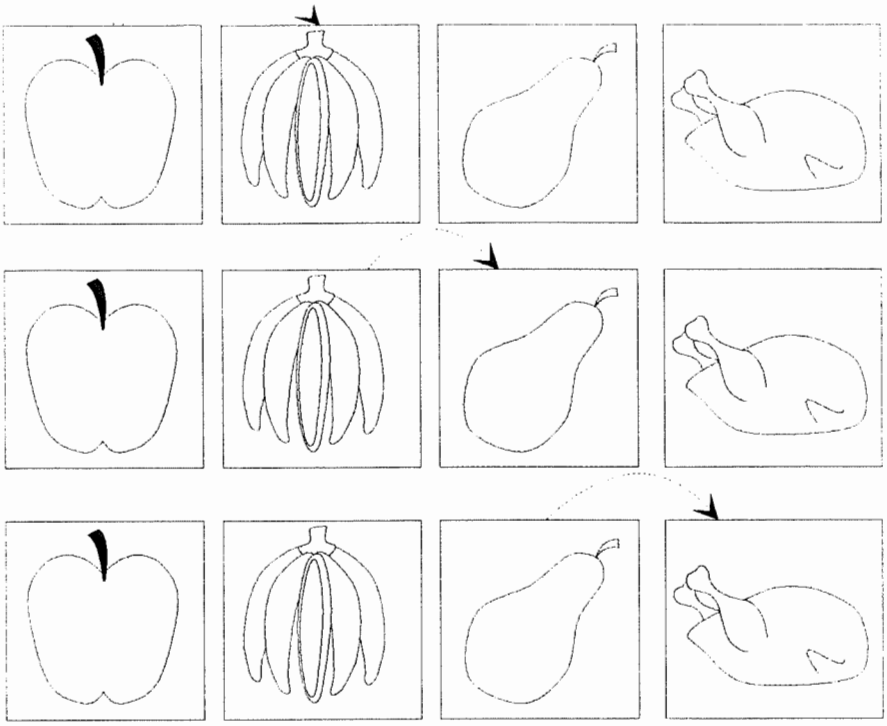


Figure 2.9. : Exercice de sélection d'un intrus.

Soit cette personne a les capacités d'utiliser la souris, il lui suffit alors de cliquer sur un des pictogrammes de son choix. Soit l'utilisateur n'a pas la capacité d'utiliser la souris. Il emploiera par exemple, un **interrupteur** (en anglais, switch) de type simple. Ce type de moyens d'interactions ne permettant que d'effectuer des validations, ce sera alors le logiciel qui devra permettre à l'utilisateur d'effectuer un choix. Pour cela le SDIU devra mettre en évidence, un par un, les divers pictogrammes qui constituent les différents choix offerts. Lorsque l'objet graphique désiré, en l'occurrence le poulet, est mis en évidence par le SDIU, il ne reste plus à la personne handicapée qu'à valider en appuyant sur son interrupteur. Nous traiterons de manière plus approfondie le problème du paramétrage de l'interface vis-à-vis du moyen d'interaction utilisé, dans le chapitre 7.

Tertio, s'il est vrai que, sans langage de programmation, les traitements effectués ne pourront pas être complexes, nous en prenons notre parti. Le logiciel Plocka, examiné à l'ACE centre d'Oxford, nous a montré qu'il ne faut pas nécessairement des centaines de fonctionnalités différentes pour réaliser des outils efficaces dans le domaine du handicap. Nous tentons, en réalisant les objets graphiques, de montrer qu'avec peu de fonctionnalités mais bien ciblées, il est possible, moyennant un peu d'imagination, d'élaborer une quantité considérable d'exercices.

Chapitre 3: Choix d'une méthode de développement orientée objet

Introduction

Tout développement informatique nécessite l'utilisation d'une méthode. Nous allons, dans le présent chapitre, justifier le choix d'une méthode de développement orientée objet (O.O.), expliquer brièvement ce qu'est une telle méthode et présenter la méthode de conception orientée objet utilisée dans le reste du travail.

3.1.Pourquoi une méthode de développement O.O.?

Lorsque nous conçevons une application, nous voulons bien évidemment que cette dernière soit de qualité. Bien qu'il soit difficile de donner une définition précise des qualités d'une application, nous proposons ici les qualités principales d'une application telles qu'elles sont présentées par B.Meyer [Meye,88].

Ces diverses qualités sont :

- ***Correction*** : capacité d'exécuter les tâches telles qu'elles sont décrites dans les spécifications.

- ***Robustesse*** : capacité de l'application de fonctionner dans des conditions anormales d'utilisation.

- ***Extensibilité*** : facilité avec laquelle l'application peut être adaptée aux changements de spécification.

- ***Réutilisation*** : capacité qu'a une application d'être réutilisée en tout ou en partie par une nouvelle application.

- ***Compatibilité*** : facilité avec laquelle l'application peut être combinée avec d'autres applications.

Pour développer une application extensible, réutilisable et compatible, il faut une méthode de conception permettant de construire une architecture flexible et décentralisée, faite d'un ensemble de modules cohérents connectés par des interfaces bien définies [Parn,84]. Une des approches, la plus appropriée dans certains contextes selon B. Meyer, permettant la conception d'une telle architecture est l'approche orientée objet.

Pour l'informaticien concepteur d'applications à interfaces graphiques, un système sera appréhendé comme une machine à états. L'état de la machine est constitué des informations stockées et de celles présentes à l'écran à un moment donné [Meye, 88][Cout, 91].

Les actions de l'utilisateur devront entraîner le passage d'un état cohérent à un autre. Elles seront effectuées sur des concepts.

Ces concepts seront matérialisés par des **objets** graphiques. La représentation des objets à l'utilisateur variera en fonction de l'état et du contexte. En outre, les actions effectuées sur les objets ne pourront modifier l'état de façon cohérente que si un certain nombre d'**opérations** leur sont attachées et sont agencées de façon adéquate.

La tâche de l'informaticien sera donc de concevoir les objets et les opérations utiles à son application avant de se lancer dans l'analyse des interactions entre opérations pour assurer la cohérence du système.

L'utilisation d'une méthodologie O.O. dans un tel contexte semble naturelle.

De plus, les objets graphiques composant l'interface de toute application interactive destinée à des personnes possédant un handicap, s'ils sont vus à un certain niveau d'abstraction, présentent des similitudes. L'utilisation d'une méthodologie O.O. permettra l'abstraction nécessaire à leur détermination. Certains objets spécifiques aux applications graphiques interactives destinées à des personnes handicapées pourront être analysés. Ils constitueront les outils de base dont il est question dans le chapitre précédent et seront implémentés sous forme de primitives.

Notons encore que ces primitives seront élaborées sous l'environnement Windows. Cet environnement encourage un type de programmation O.O.. C'est aussi pour cette raison que nous utiliserons une méthodologie O.O. lors du processus de conception.

3.2.Qu'est-ce qu'une méthode de conception orientée objet?

Cette section n'a pour ambition que de donner quelques éléments généraux relatifs aux méthodes de conception orientées objets. Ces éléments sont nécessaires à la compréhension de la méthode de conception exposée et utilisée dans la suite du travail. L'abondance et la complexité de la littérature dans ce domaine montre qu'une analyse approfondie aurait dépassé largement le cadre de ce travail. Le lecteur intéressé trouvera dans [Cox, 86], [Booc, 86], [Khos,90], [Meye, 88] de plus amples informations.

3.2.1. La méthode

Une **méthode de conception orientée objet** est une méthode qui se base sur les données d'une application plutôt que sur les fonctions de celle-ci.

Dans le cadre du cycle de vie d'une application, les fonctions évoluent au cours du temps alors que la structure des données qu'elles manipulent ne changent pratiquement pas.

"Une **application** est un modèle opérationnel de certains aspects du monde" [Meye, 88]. Elle est un "modèle" car elle est basée sur une interprétation de certains phénomènes du monde et elle est "opérationnelle" car elle génère des résultats pratiques qui sont rendus au monde.

Lorsqu'une application est vue comme un modèle opérationnel du monde, l'approche orientée objet paraît naturelle. En effet, nous pouvons voir le monde à modéliser comme constitué d'un ensemble d'objets. Il nous paraît dès lors naturel d'organiser le modèle autour des représentations de ses objets dans le système.

Les objets, lorsqu'ils ont les mêmes propriétés formelles, appelées **attributs**, et les mêmes comportements sont regroupés en classes d'objets. Une classe d'objets est donc une spécification générique d'un nombre quelconque d'objets similaires.

Une **classe d'objets** est décrite par les services (appelés aussi fonctions, opérations ou méthodes dans la littérature) disponibles sur cette classe et les propriétés formelles des objets de la classe. Elle est composée d'une partie publique et d'une partie privée. La **partie publique** définit les services qu'elle peut exécuter : les manipulations d'attributs ainsi que les autres services qu'elle peut fournir. En d'autres termes, la partie publique décrit toutes les interactions possibles entre l'objet et le monde extérieur. La **partie privée** est constituée des propriétés formelles des objets de la classe et de l'implémentation des services. Car la façon dont on effectuera ces derniers ne concerne en aucune façon le reste du système. De plus, certains d'entre eux peuvent être privés s'ils ne sont pas jugés pertinents pour l'extérieur.

Les objets communiquent entre eux par l'envoi de **messages** via les parties publiques des classes d'objets.

Une **instance ou occurrence d'une classe** est un représentant particulier de la classe. Il aura les mêmes propriétés formelles et les mêmes comportements que les autres représentants mais possédera des valeurs particulières de ses attributs.

Ce type de conception conduira à une architecture dont les modules seront les classes d'objets. Ces derniers utiliseront les services offerts par les autres modules grâce à leurs parties publiques et seront assemblés selon la méthode Bottom-Up.

Remarque : nous constatons souvent que, les applications sont un compromis entre l'approche orientée objet et l'approche orientée fonction. C'est-à-dire que leur architecture est constituée d'un ensemble de modules de bas niveau qui constituent les classes d'objets décrites ci-dessus, alors que les modules de niveaux supérieurs sont des modules fonctionnels assemblés selon la méthode Bottom-up. Ces derniers utilisent les services des modules de niveaux inférieurs.

3.2.2.Quelques mécanismes à la base d'un langage de conception O.O.

Nous présenterons ici, les mécanismes qui sont à la base du principe de fonctionnement de la conception O.O..

A.L'encapsulation

Mécanisme par lequel des connaissances et des services qui ont des points communs au niveau conceptuel sont rassemblés.

B.Le masquage de l'information

Mécanisme par lequel une partie de l'information est cachée vis à vis de l'extérieur.

Ces deux mécanismes sont utilisés lors de la création d'une classe. En effet, comme indiqué précédemment, une classe reprend un ensemble de propriétés formelles et de services. Ceux-ci sont encapsulés dans la classe et permettent ainsi une certaine abstraction. D'autre part, seule la partie publique, appelée aussi **interface**, de la classe sera connue du reste du monde. Toute la partie implémentation sera masquée. Ce mécanisme permet une certaine réduction de la complexité du problème lors de la conception. En effet, tous les problèmes liés à l'implémentation des services seront résolus à l'intérieur d'une classe de façon transparente pour les concepteurs des autres classes. De même, lors des changements à effectuer après la réalisation de l'application (extension ou correction), le concepteur devra s'occuper de la partie implémentation uniquement.

Outre ces avantages, ces deux mécanismes ouvrent le champs à la réutilisation de classes.

C.L'héritage

L'héritage est un mécanisme grâce auquel une classe définit les services et les structures de données de ses instances comme un sur-ensemble de la définition d'une ou de plusieurs autres classe(s). La classe ainsi définie est une spécialisation de l' (des) autre(s).

La classe qui hérite est appelée sous-classe alors que la classe dont elle hérite est la super-classe.

Si une classe possède plusieurs super-classes, nous parlerons d'héritage multiple. Une sous-classe peut hériter de l'interface, du code d'implémentation et/ou des attributs de sa (ses) super-classe(s).

Tous ces mécanismes ne peuvent être réellement obtenus que grâce à un langage permettant l'**overloading** et la **généricité**.

D.L'overloading

L'overloading consiste en l'utilisation d'un même nom pour plus d'une spécification (typiquement, des procédures différentes mais ayant le même nom). Cette facilité syntaxique permettra un masquage de l'information plus important.

En effet, l'overloading permettra à deux classes de répondre différemment à un même message. La façon dont elles s'y prendront sera transparente pour l'utilisateur du service.

E.La généricité

La généricité est la "capacité de définir des modules paramétrés" [Meye, 88]. Ces modules possédant des paramètres formels ne seront pas utilisables directement mais plutôt en leur fournissant des types effectifs pour les paramètres effectifs.

On utilisera des classes d'objets comme types effectifs. Ainsi, un même service pourra être appliqué à des classes distinctes. Les services ainsi définis permettront de constituer des classes abstraites. Celles-ci existent pour que les services communs à une variété de classes puisse être extraits et placés en un endroit accessible par chacune.

3.3.La méthode O.O. utilisée : OBLOG

Nous allons, à présent, caractériser la méthode de conception O.O. utilisée dans le reste du travail. Nous éclairerons d'abord le lecteur sur ses inventeurs et leurs ambitions et présenterons ensuite les concepts nécessaires à son utilisation dans le cadre du mémoire. Cette section est basée sur les documents réalisés par deux membres du personnel responsables de la réalisation et de la promotion du produit [Zeip, 92].

3.3.1.Les inventeurs et leurs ambitions

Les concepts sous-jacents à OBLOG ont été définis par Monsieur le Professeur P. Sarnadas. OBLOG ou OBject-oriented LOGic est toujours en cours de développement par le personnel de l'ESDI.

Leur mission est de développer une nouvelle génération de systèmes d'informations financiers pour le holding financier portuguais Grupo Esperito Santo (GES). Pour y parvenir, ils développent leur propre outil, un atelier logiciel (en anglais, CASE Workbench) appelé OBLOG et l'utiliseront pour développer les différents composants de leur système.

L'ambition de l'approche OBLOG est l'utilisation d'un langage de spécification unique, formel, orienté objet et pictographique à travers le cycle de vie du développement. Ce langage sera supporté par un environnement adéquat.

En effet, le même langage sera utilisé pour l'analyse, la conception, le codage, les essais et la maintenance ceci afin de permettre de combler le fossé sémantique existant habituellement entre les différentes étapes d'un projet. Les acteurs agissant à ces différents niveaux contribueront tous au produit final. En particulier, cela permettra de réduire l'espace existant habituellement entre les spécifications et les programmes puisque le code sera généré automatiquement.

Cependant, pour être utilisable, le langage OBLOG doit être simple et relativement intuitif pour pouvoir être utilisé par des personnes sans qualification formelle, par exemple, lors de la détermination des besoins d'un client. Le caractère graphique du langage aidera dans la réalisation de cet objectif. Il doit, de plus, être de haut niveau pour permettre de représenter le monde réel et doit être puissant pour pouvoir résister aux différentes étapes du développement.

En outre, le fait que le langage soit orienté objet permettra d'atteindre les qualités d'un logiciel telles qu'elles ont été exposées au début de ce chapitre.

La puissance des concepts sous-jacents à l'approche rendent l'utilisation de la méthode et du langage particulièrement attrayante pour l'analyse et la conception d'une application.

3.3.2. Introduction au langage OBLOG

Le langage OBLOG est très général. Il est basé sur un grand nombre de concepts relatifs aux méthodes O.O. pour le développement de grosses applications. Nous n'utiliserons pas tous les concepts disponibles pour les besoins du travail. En conséquence, nous ne présenterons que les concepts utilisés afin d'éviter d'alourdir inutilement le travail.

Selon l'approche OBLOG, un objet peut être défini comme un ensemble d'attributs et d'événements qui peuvent affecter les valeurs de ces attributs. Des règles de mise à jour d'attributs sont associées aux événements. En outre, un objet est une occurrence d'une classe d'objets. Nous pouvons remarquer, au niveau de la méthode, que les attributs sont les propriétés formelles, alors que les événements sont les services dont il est question au début du présent chapitre.

Un objet est aussi caractérisé par son comportement. Le comportement d'un objet est la déclaration des séquences possibles d'événements et leurs conditions de déclenchement.

Certains événements sont passifs, c'est-à-dire qu'ils sont déclenchés par une demande externe à l'objet, ils peuvent donc être vus comme des services offerts. D'autres sont actifs, déclenchés par l'objet lui-même, ils peuvent être vus comme des services internes. Ces propriétés permettent de représenter la dynamique d'enchaînement des événements.

Le mécanisme standard utilisé pour la communication entre objets est l'appel d'événements. L'événement appelé doit être passif alors que l'événement appelant peut être actif ou passif (s'il est lui-même appelé par un autre). La communication peut avoir lieu entre deux objets d'une même classe ou entre deux objets de classes différentes.

3.3.3. Les concepts OBLOG et leurs représentations

Les objets et les classes d'objets :

Une classe d'objets est un ensemble d'objets "similaires" appelés instances de la classe.

Un objet contient des attributs et des événements. Il est caractérisé par son comportement et par son état. L'état d'un objet est représenté par les valeurs de ses attributs à un moment donné.

Une classe, quant à elle, est décrite par :

- son diagramme matriciel,
- le diagramme de son comportement,
- les initialisations des valeurs de ses attributs et les mises à jour de ces derniers,
- les interactions avec les autres classes.

Une classe d'objets est marquée par "?" lorsqu'elle peut contenir plusieurs instances. Elle est marquée "1" lorsqu'elle est constituée d'une seule instance.

Exemple de représentation :

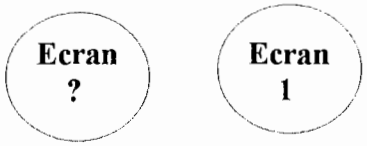


Figure 3.1 : Représentation de classes

L'attribut :

L'attribut prend ses valeurs dans un codomaine défini par un type de donnée ou une classe d'objets.

Un attribut peut être constant ou variable : un attribut est constant lorsque sa valeur est fixée à la naissance de l'objet et n'est pas modifiée durant le cycle de vie de l'objet, il est variable lorsque sa valeur change durant ce dernier.

Il peut être total ou partiel : un attribut est total lorsqu'il doit avoir une valeur à tout moment durant le cycle de vie d'un objet, il est partiel s'il peut être indéfini à un certain moment du cycle de vie d'un objet (un attribut partiel est variable).

Un attribut clef est utilisé pour distinguer les instances d'une classe. Ce type d'attribut ne sera donc nécessaire que pour les classes marquées par "?". Seules les valeurs de l'attribut clef peuvent être utilisées au-delà de la portée d'un objet.

Le nom d'un attribut est local à une classe. Sa valeur peut être dérivée d'autres attributs.

Exemple de représentation:

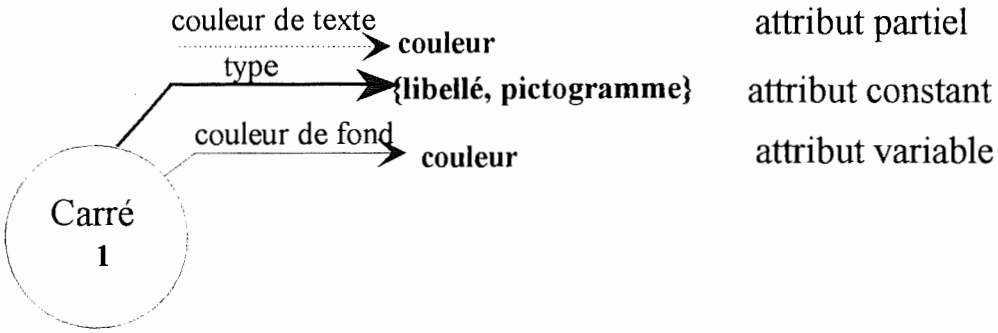


Figure 3.2 : Représentation des attributs d'une classe

Cet objet sortant tout droit de notre imagination est un carré qui peut être illustré soit par un libellé, soit par un pictogramme mais lorsqu'il est créé, il ne peut changer de type. S'il est un pictogramme, il n'aura évidemment pas de couleur de texte. Cet objet peut en outre changer de couleur de fond pendant sa vie.

Le type de donnée :

Un type de donnée peut être prédéfini ou défini par le concepteur.

Les types de données prédéfinis sont :

- type simple : booléen, naturel, entier, caractère,...
- type paramétrisé : liste (t), pile (t),...
- type constructeur : énuméré,...

Les types de données définis par le concepteur le sont à partir des types prédéfinis. Ils pourront être des types structurés :

- Seq [type] sera une suite d'éléments d'un type prédéfini ou défini par l'utilisateur.
- CP [type1, type2] sera un tuple d'éléments d'un type prédéfini ou défini par l'utilisateur.
- Set [type] sera un ensemble d'éléments d'un type prédéfini ou défini par l'utilisateur.

L'événement et la classe d'événements

Un événement est une "action atomique" qui affecte la vie et/ou l'état d'un objet. De la même façon que l'on parle d'objet et de classe d'objets, une classe d'événements est une spécification générique d'un nombre quelconque d'événements similaires.

Une classe d'événements dispose d'attributs d'événements. Le codomaine des attributs d'événement peut être un type de donnée, une classe d'objets ou une classe d'événements.

La vie d'un objet commence avec un événement de naissance, suivi d'un certain nombre d'événements de mise à jour et se termine, en général, par un événement de mort. Un événement de naissance est marqué par "*". Un événement de mort par "+".

Un événement peut être actif ou passif : un événement d'un objet est actif lorsqu'il est déclenché par cet objet. Un objet contenant au moins un événement actif est actif. Un événement actif est marqué par "!". Un événement d'un objet est passif lorsqu'il est déclenché par un autre objet. Un objet ne contenant pas d'événement actif est passif. Lorsqu'un événement est réalisé par l'"extérieur", hors du système d'information décrit, il est marqué par "°".

Le nom d'une classe d'événements est local à la classe d'objets.

Le nom d'un attribut d'événement est local à la classe d'événements.

Exemple de représentation :

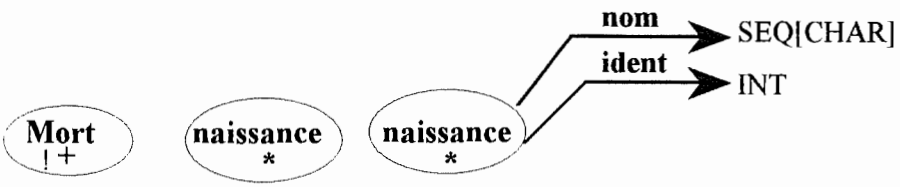


Figure 3.3 : Représentation des événements et de leurs attributs

Ces trois événements représentent la mort de l'objet auquel l'événement sera rattaché, la naissance d'un objet et la naissance d'un objet auquel un nom et un identifiant seront attachés. Le nom sera représenté par un string (suite de caractères) et l'identifiant par un entier. L'événement de mort est actif, c'est-à-dire que l'objet décide de sa mort.

La description d'un objet :

La description d'un objet est réalisée à l'aide du diagramme matriciel de la classe d'objets à laquelle il appartient. Il contient la définition des attributs et des événements des objets constituant la classe d'objets ainsi que la définition des attributs d'événements.

Exemple de représentation :

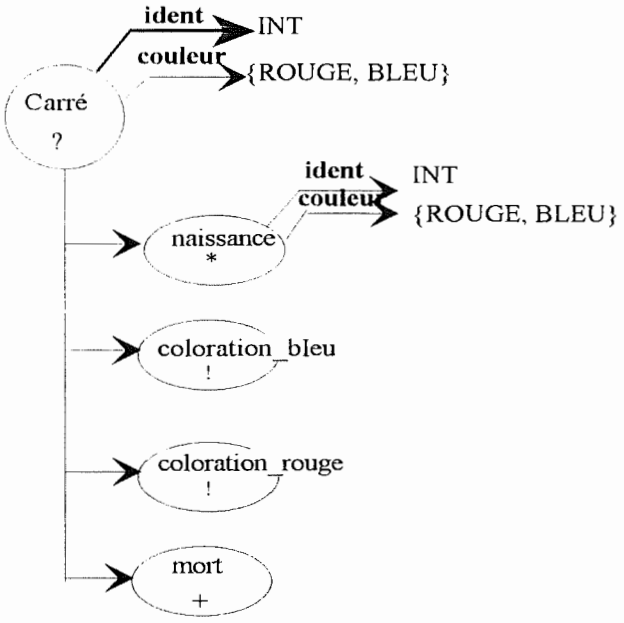


Figure 3.4 : Diagramme matriciel d'une classe d'objets

Cette classe d'objets est appelée **Carré**. Les événements qui lui sont attachés sont **naissance**, **mort**, **coloration_bleu** et **coloration_rouge**. Les événements de coloration sont déclenchés par l'objet lui-même; ils sont actifs. **Naissance** est le seul événement possédant des attributs. Chaque objet de la classe possédera en outre des valeurs d'attributs spécifiques.

Les initialisations de valeurs d'attributs et les mises à jour :

Les événements de naissance doivent initialiser les valeurs des attributs constants et peuvent initialiser celles des attributs variables.

Exemple de représentation :

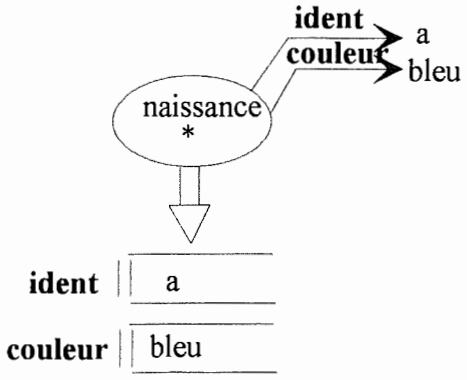


Figure 3.5 : Les initialisations d'attributs

Ce diagramme représente les initialisations des attributs des objets de la classe. Chaque carré de la classe aura un identifiant et sera coloré en bleu lors de sa naissance.

Les événements de mise à jour ne peuvent modifier que les valeurs des attributs variables.

Exemple de représentation :

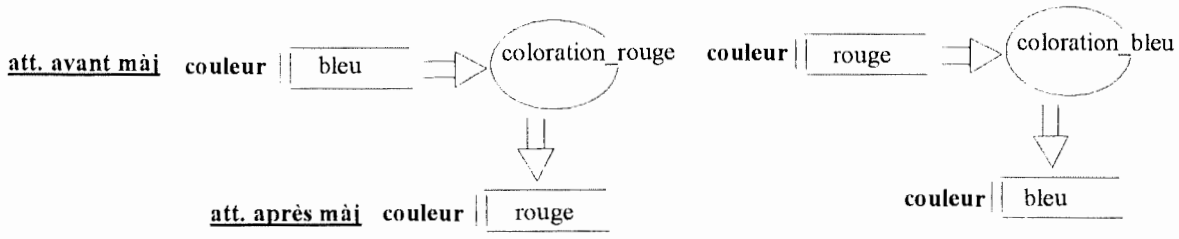


Figure 3.6 : Les mises à jour d'attributs

Le comportement d'un objet :

Le comportement d'un objet est constitué des séquences d'événements possibles constituant le cycle de vie de l'objet.

Tout cycle de vie d'un objet débutera avec un événement de naissance, suivi de 0, un ou de plusieurs événements de mise à jour. Il se terminera, en général, par un événement de mort.

Un événement permet le passage d'une situation à une autre.

Exemple de représentation :

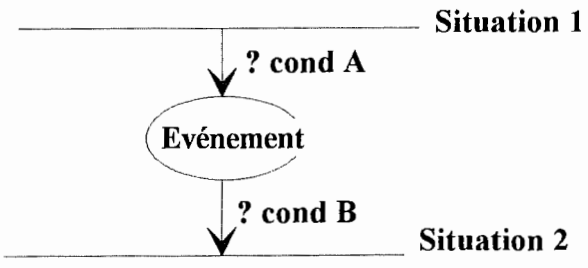


Figure 3.7 : Un événement dans un diagramme de comportement

Il est possible de définir des pré-conditions et des post-conditions sur les événements. Celles-ci contraignent le déclenchement de l'événement. L'événement de la

figure 3.7 ne pourra avoir lieu que lorsque dans la situation 1, la condition A est vérifiée et si la situation 2 remplit une autre condition B.

Le comportement des objets est décrit par le diagramme de comportement de leur classe.

Exemple de représentation :

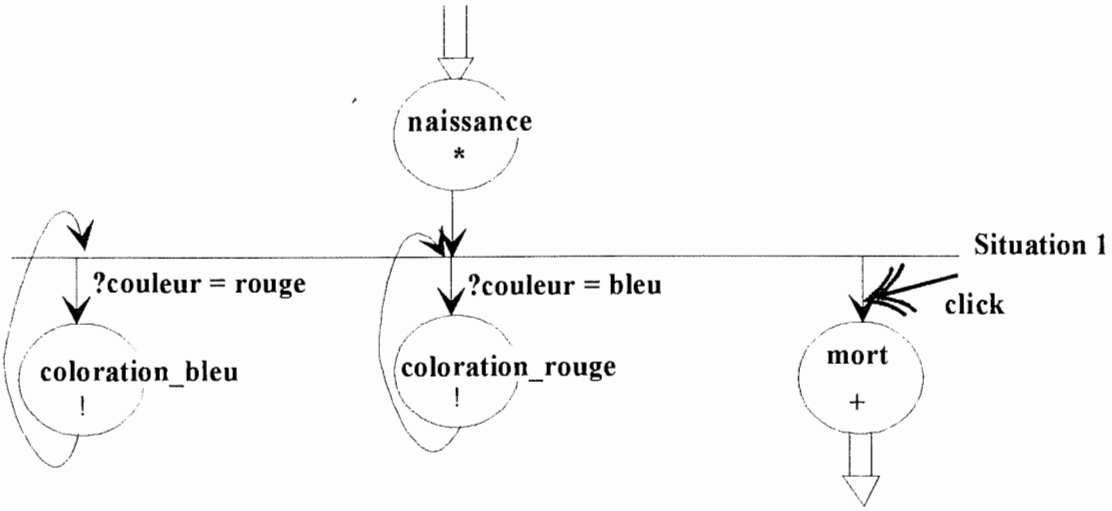


Figure 3.8 : Le diagramme de comportement

Ici, après sa **naissance**, le carré se trouve dans la situation 1. Nous savons, grâce aux diagrammes d'initialisation et de mise à jour, que le carré se trouvant dans cette situation a un identifiant et est coloré. On peut déclencher une occurrence de l'événement **coloration_bleu** ou de l'événement **coloration_rouge** suivant que la condition couleur = rouge ou couleur = bleu est vérifiée. Quel que soit l'événement déclenché, l'objet reviendra dans la même situation. Un événement externe déclenche la **mort** de l'objet; c'est l'interaction avec l'utilisateur lorsqu'il sélectionne l'objet grâce à un moyen d'interaction.

Les interactions entre objets

Chaque objet vit dans une communauté d'objets. A ce titre, il interagit avec les autres objets de la communauté. Ces interactions peuvent être réalisées par la communication entre objets ou par le partage d'éléments d'objets.

A.La communication entre objets

La communication entre objets est réalisée par l'appel d'événements.

Un événement a d'une classe d'objets A peut appeler un événement b d'une classe d'objets B (ou A). Dans ce cas, quand l'événement a d'une instance de la classe d'objets A est déclenché, l'événement b sera déclenché dans le chef de l'instance de la classe B (A) correspondante. Seuls les événements passifs peuvent être appelés. Les événements appelants, quant à eux, peuvent être actifs ou passifs. Nous pouvons, en outre, imposer des contraintes aux appels d'événements (contraintes sur les valeurs d'attributs de l'appelant, par exemple).

L'appel d'événements et le passage éventuel de valeurs qui peut y être associé est décrit par le diagramme des interactions entre classes d'objets.

Exemple de représentation :

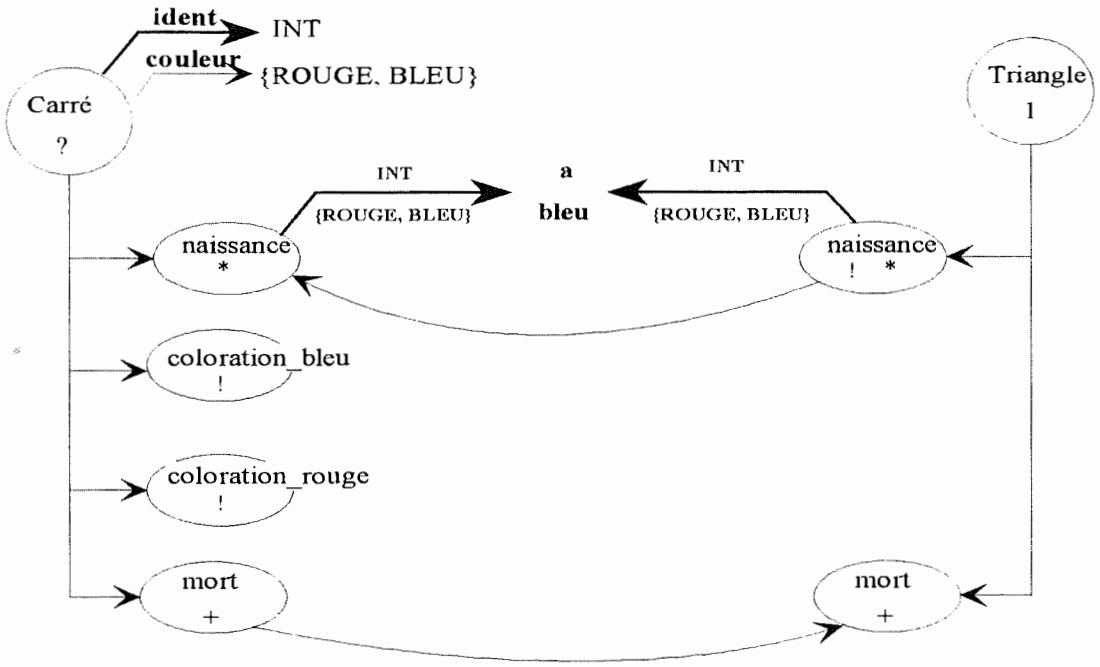


Figure 3.9 : Le diagramme d'interactions

Ici, un triangle naît automatiquement (car son événement de naissance est actif). Lorsqu'il naît, il provoque la naissance d'un carré d'**identifiant** "a" et de **couleur** "bleu". Ce dernier se comporte comme indiqué précédemment. Lorsque le carré meurt, il provoque la mort du triangle.

B.Le partage d'éléments d'objets

Le partage d'éléments d'objets peut être le partage d'attributs et/ou d'événements. Il sera réalisé, entre autres, par l'***abstraction***.

Cette dernière permet de construire des interfaces d'objets en extrayant certains aspects d'une classe d'objets. L'abstraction est stricte si l'interface contient tous les événements exprimant les initialisations et les mises à jours des attributs retenus. Elle est libre si certains attributs changent de valeur sans explication locale.

Chaque instance de l'interface interagit avec l'instance correspondante de la classe dont elle est abstraite. En effet, chaque événement de la première est un événement de la deuxième et une valeur d'attribut de l'instance d'interface doit être une valeur du même attribut de l'instance de l'objet.

Exemple de représentation dans le langage :

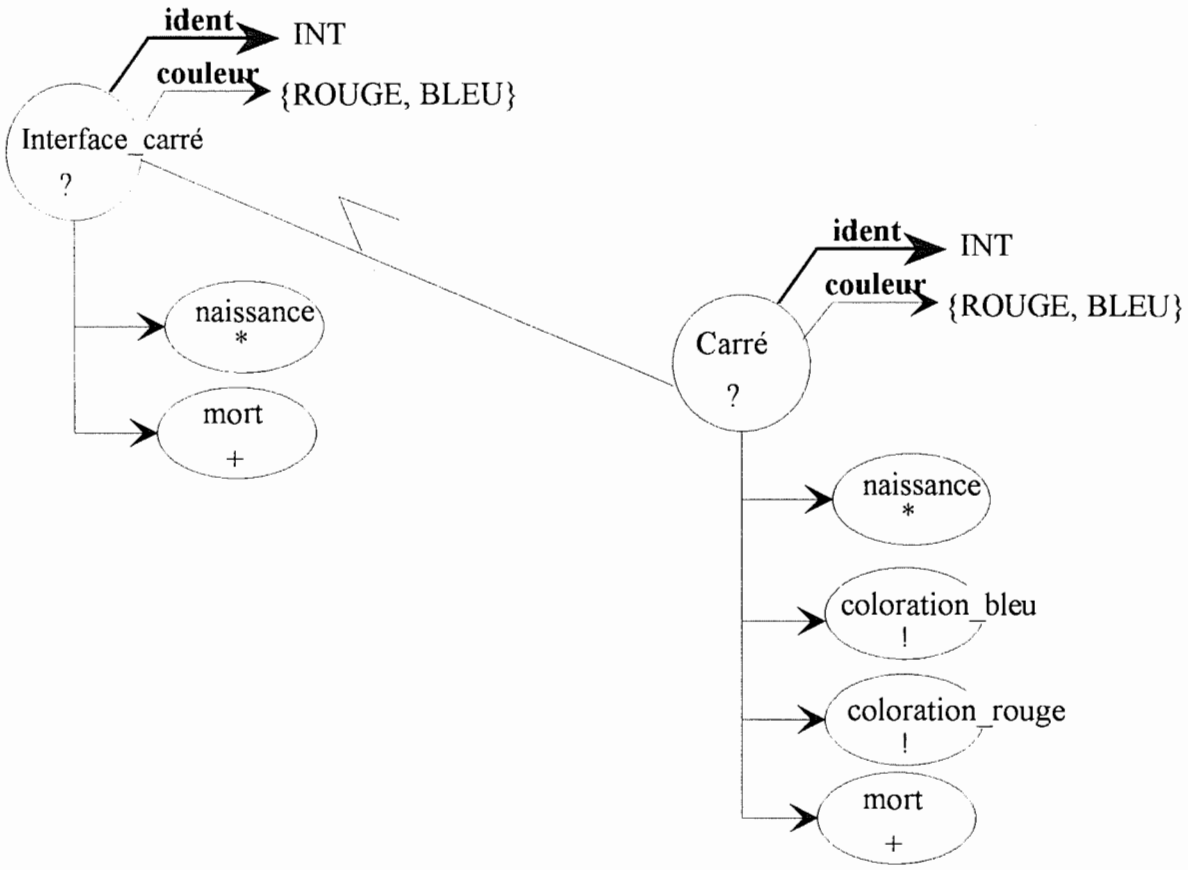


Figure 3.10 : L'abstraction

Cette figure est un exemple d'abstraction libre. En effet, la valeur de l'attribut **couleur** est modifiée par les événements de coloration qui ne font pas partie de l'**interface_carré.**

Chapitre 4 : Choix et analyse des objets interactifs

Introduction

G.Warnant [Warn, 88] indique qu'une application interactive est composée de deux parties principales : le dialogue et les fonctions de l'application.

La partie ***dialogue*** est chargée de gérer la communication avec l'utilisateur alors que les ***fonctions de l'application*** traitent les données reçues pour réaliser les fonctionnalités définies.

Selon T. Lepoutre et J-M. Roquet [Lepo, 90], le caractère hautement interactif des applications pour personnes handicapées a amené les fonctionnalités à perdre toute substance si elles sont dissociées de l'interface. "Le service offert à l'utilisateur est finalement l'interface proprement dite... l'application pour personne mentalement déficiente n'est qu'un dialogue continu".

La structure du dialogue, quant à elle, peut être scindée en deux parties : la ***tâche*** et ***l'interface concrète*** [Warn, 88].

La tâche est l'expression des actions que l'utilisateur doit réaliser pour mener à bien l'exécution de l'application interactive. Une tâche est composée de messages interactifs, d'opérations définies sur chaque message et de règles d'enchaînement des opérations.

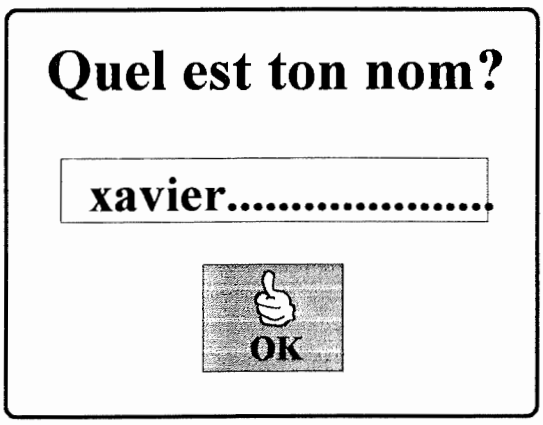
L'interface concrète, quant à elle, est la présentation de la tâche, c'est la partie visible à l'utilisateur. Elle est composée des objets interactifs concrets (fenêtre, icône,...) qui sont la matérialisation des messages interactifs et sur lesquels l'utilisateur peut effectuer des actions physiques (cliquer au moyen de la souris sur un objet, remplir un champ d'édition au moyen du clavier,...). Elle est également constituée des contraintes d'enchaînement de ces actions.

L'objet du logiciel est d'offrir à l'éducateur réalisant de petits exercices pour personnes handicapées la possibilité de se concentrer sur la tâche et de passer moins de temps à la réalisation de l'interface concrète. Il sera aidé dans la réalisation de celle-ci par l'ensemble des outils et des composants du logiciel qui seront décrits au chapitre 7. Notre travail consiste à offrir les outils permettant de réaliser les objets interactifs concrets nécessaires aux exercices interactifs.

Les objets interactifs concrets dépendent du contexte de la tâche puisqu'ils sont la matérialisation des messages interactifs de celle-ci. Illustrons cette affirmation par un exemple. Dans le logiciel "Ordithéâtre" réalisé dans le cadre d'un mémoire à l'Institut [Brou, 90], le concepteur demande à l'utilisateur de donner un nom à la scène qu'il a créée. Voici l'objet interactif concret correspondant au dialogue :

Nous ne pouvons évidemment pas offrir cet objet particulier tel qu'il est puisqu'il faudrait créer autant d'objets différents qu'il existe de libellés d'identification du champ d'édition.

Nous avons vu, au chapitre 3, qu'une méthode orientée objet fournissait l'abstraction nécessaire à la détermination de classes d'objets. Ces classes d'objets décrivent les structures et les comportements communs à un ensemble d'objets. Cette méthodologie semble tout à fait appropriée au problème qui nous occupe. En effet, dans notre exemple, l'objet interactif concret présenté dans le logiciel Ordithéâtre aura la même structure et les mêmes comportements que cet autre objet interactif concret rencontré dans un autre logiciel :



Réussir à déterminer et à implémenter la classe d'objets correspondante signifie que tout concepteur pourra utiliser ce type d'objet sans avoir à le concevoir et à l'implémenter à nouveau.

Cette méthode nous conduit à établir une description abstraite des objets interactifs concrets composant l'interface concrète d'un certain nombre d'applications classiques pour personnes présentant un handicap.

Ces classes d'objets interactifs seront instanciées par le concepteur en fonction des caractéristiques des utilisateurs et de la tâche.

Ces classes ont été déterminées d'une part par l'analyse des objets interactifs classiques, et d'autre part par l'analyse des objets interactifs généralement utilisés dans les mémoires précédemment réalisés dans le domaine du handicap et présentant une "bonne reconnaissance" de la part des utilisateurs. Elles respectent en outre les critères généraux guidant la conception d'une interface [Shne, 87].

Nous indiquerons d'abord quelles sont les classes d'objets offertes, leurs comportements et les critères ayant motivé leur choix. Nous présenterons ensuite l'analyse conceptuelle des objets en OBLOG.

4.1.Les objets interactifs

Nous présenterons dans cette section la liste des objets interactifs qui seront nécessaires aux concepteurs d'applications interactives pour personnes handicapées.

Voici la liste des objets interactifs que nous avons choisi d'offrir :

- la liste fixe,
- le dérouleur,
- la boîte de dialogue,
- la boîte d'édition,
- l'écran,
- l'icône.

Pour chacun d'entre eux, nous donnerons une brève explication de leur comportement et des raisons nous ayant poussés à les offrir.

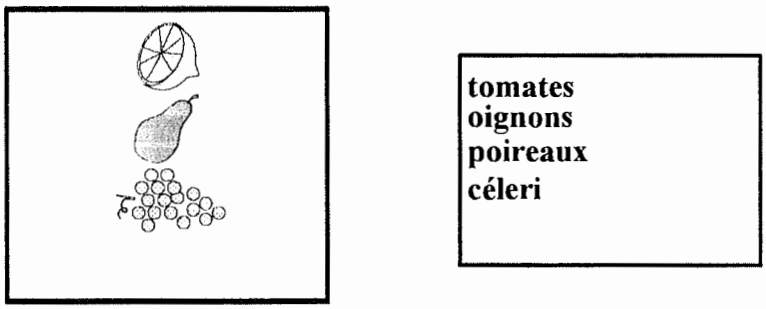
4.1.1.La liste fixe

Nous avons indiqué, dans le premier chapitre, qu'un des dilemmes se présentant au concepteur d'une application est le dilemme de la quantité d'informations à disposer à l'écran. Les personnes handicapées possédant un faible niveau de compréhension et des

difficultés au niveau de la rétention des informations, il faudra une interface plus riche et mieux adaptée quant à son contenu informatif.

Le problème de la pertinence du contenu informatif relève du concepteur de l'application puisqu'il est fonction du contexte de la tâche. Nous devons cependant donner la possibilité au concepteur de présenter à l'écran l'information jugée nécessaire et suffisante à la compréhension de la tâche par l'utilisateur.

La ***liste fixe*** est un objet graphique permettant d'afficher à l'écran une liste d'éléments. Les éléments de la liste seront soit des libellés, soit des pictogrammes. Cet objet n'autorisera aucun mécanisme de défilement.



Il permettra à l'utilisateur de l'application interactive de réduire l'effort de mémorisation des informations relatives à la tâche.

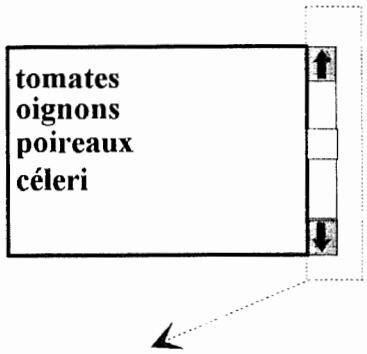
Il offre aussi un renforcement visuel du regroupement logique d'informations effectué.

Le concepteur d'applications utilisant une liste fixe pourra déterminer la couleur de fond, le nombre d'éléments et la localisation de cette dernière. Il pourra, en outre, donner les éléments de la liste. Celui qui utilise une liste fixe de texte pourra de plus déterminer la couleur du texte de la liste. Ces différentes caractéristiques lui permettront d'adapter l'objet aux besoins de l'utilisateur.

4.1.2. Le dérouleur

Lors d'un dialogue, il arrive souvent que l'utilisateur doive opérer un choix entre plusieurs éléments. Si le nombre d'éléments parmi lesquels il doit choisir est important, le concepteur est confronté à un problème. En effet, faute de place à l'écran, il ne peut présenter les différentes propositions en une seule fois.

Dans les applications graphiques interactives destinées à des personnes dites normales, un objet interactif appelé liste de sélection est souvent utilisé pour résoudre ce problème en voici une représentation:



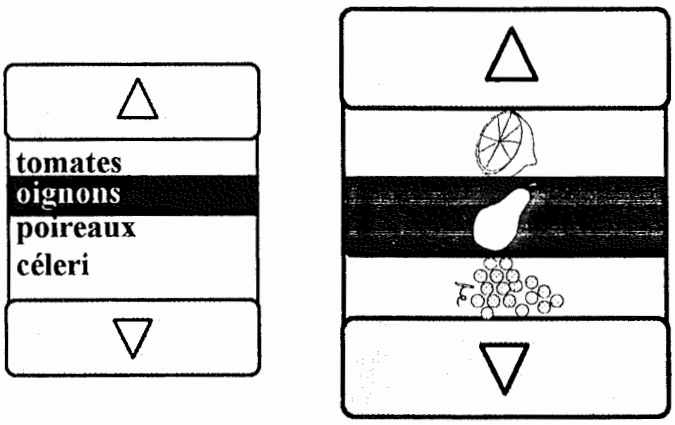
Barre de défilement

Nous ne décrirons pas le comportement de cet objet en raison de sa large utilisation dans de nombreuses applications. Nous supposons donc que le lecteur en connaît le fonctionnement. Le lecteur intéressé en trouvera une description complète dans [Prov, 91].

Nous remarquons qu'un tel objet est trop compliqué pour des personnes présentant une déficience mentale. En effet, elles auront des difficultés à comprendre le mécanisme de la barre de défilement, et à positionner correctement le pointeur de la souris sur les flèches en raison de la faible taille de celles-ci.

Nous avons donc décidé d'offrir un objet mieux adapté à la population. Cet objet permettra, comme la liste fixe, de réduire la charge de mémorisation de l'information dans le chef de l'utilisateur. Il lui donnera de plus le sentiment qu'il contrôle l'application puisqu'il peut manipuler l'objet à souhait pour effectuer son choix. Cette caractéristique est à analyser avec précaution car si elle est source de motivation chez certaines personnes, elle peut aussi augmenter le sentiment d'insécurité présent chez d'autres. En effet, la personne handicapée pose difficilement des choix. Il convient dès lors de la soutenir et de la guider pour les réaliser. Certaines d'entre elles se sentiront motivées par la liberté qui leur est offerte tandis que d'autres, plus craintives, la ressentiront comme un abandon, ce qui peut contribuer à l'inhibition intellectuelle. Le concepteur de l'application décidera ou non d'utiliser l'objet en fonction de la population à considérer.

Cet objet sera appelé ***dérouleur***. Il est en fait une liste déroulable d'éléments représentant des choix parmi lesquels l'utilisateur a la possibilité de faire une sélection. Un et un seul élément pourra être sélectionné à la fois. C'est ce que nous appellerons une sélection simple. Il est constitué d'une zone rectangulaire contenant les éléments du dérouleur présentés les uns en dessous des autres et de deux flèches de défilement. L'une située à la base de la zone rectangulaire et l'autre située au sommet de cette dernière.



Nous avons observé l'utilisation d'un objet semblable, dans un des mémoires réalisés précédemment [Tope,91]. Ses éléments étaient uniquement des libellés. Cependant, Madame le Professeur M. Noirhomme nous a justement fait remarquer qu'il était pertinent d'offrir la possibilité de définir des dérouleurs dont les éléments sont des pictogrammes. En effet, une large proportion de personnes possédant une déficience mentale ne sait pas lire.

Outre la distinction entre dérouleur graphique et dérouleur de texte, une autre classification des dérouleurs peut être réalisée en fonction de leur comportement. Nous les nommerons "dérouleur à sélection par bouton" et "dérouleur à sélection directe".

La sélection d'un élément d'un dérouleur à sélection par bouton sera effectuée en cliquant sur une des deux flèches de défilement pour la sélectionner. L'élément suivant ou précédent se verra alors contrasté afin d'indiquer qu'il est couramment sélectionné, et les éléments du dérouleur seront décalés dans la direction indiquée par la flèche sélectionnée. Nous sommes en présence d'un mécanisme de sélection simple bien que plusieurs éléments du dérouleur puissent être sélectionnés au cours du temps. Si l'utilisateur sélectionne la flèche de défilement vers le bas et que l'élément couramment sélectionné est le dernier, ou, s'il sélectionne la flèche de défilement vers le haut et que l'élément couramment sélectionné est le premier, l'élément sélectionné ne sera pas modifié et aucun décalage des éléments du dérouleur ne sera effectué.

La sélection d'un élément d'un dérouleur à sélection directe, quant à elle, sera effectuée par la personne handicapée en cliquant directement sur l'élément à l'aide de la souris. Le mécanisme de défilement sera utilisé pour examiner les différents éléments mais n'entraînera aucune sélection de ces derniers. Si l'utilisateur sélectionne la flèche de défilement vers le haut et que le premier élément du dérouleur est visible ou s'il sélectionne la flèche de défilement vers le bas et que le dernier élément est visible, aucun décalage d'élément ne sera effectué.

Nous offrirons donc quatre types de dérouleurs différents : le dérouleur de texte à sélection par bouton, le dérouleur de texte à sélection directe, le dérouleur graphique à sélection par bouton et le dérouleur graphique à sélection directe.

Nous avons décidé, lors de la création d'un dérouleur à l'écran, de présenter son premier élément contrasté mais vide. Ceci afin d'éviter que la personne handicapée ne soit perturbée par un élément contrasté d'avance. Cet élément est insélectionnable.

Il est évidemment déconseillé au concepteur de mélanger les dérouleurs à sélection par bouton et les dérouleurs à sélection directe dans une même application. Ceci ne manquerait pas de troubler l'utilisateur.

Lorsque le concepteur de l'application interactive décrira un dérouleur, il aura la possibilité de déterminer la taille des flèches de défilement, leur couleur de fond, la couleur de leur libellé, le type des éléments du dérouleur, la couleur de fond de ces derniers. La couleur du texte, la couleur du texte d'un élément lorsqu'il est sélectionné et la couleur de fond à utiliser lorsqu'un élément est contrasté sont à déterminer pour les éléments d'un dérouleur de texte. Il faudra aussi donner la localisation du dérouleur et par là sa taille et le nombre total d'éléments. Autant de caractéristiques permettant d'adapter le dérouleur à l'utilisateur.

4.1.3.La boîte de dialogue

Dans des applications destinées à des personnes dites normales, il est important d'offrir des retours d'informations à l'utilisateur. Ils lui permettent d'ailleurs, dans de nombreux cas, d'éviter de faire des erreurs ou de les corriger. Cette importance est encore accrue si l'utilisateur présente une déficience mentale.

Nous avons indiqué précédemment que les personnes présentant un hancidap ont une rationalité, une logique différente de la nôtre. Nous voudrions cependant leur permettre de réaliser différentes tâches avec l'outil informatique. Pour y arriver, il faudra les guider. Le dialogue sera essentiellement dirigé par les besoins de l'application tandis que l'utilisateur sera essentiellement réactif. En conséquence, nous avons choisi d'offrir un type de boîte de dialogue modale. Ce type de boîte empêche l'utilisateur de continuer à réaliser la tâche tant qu'il n'a pas agi dans la boîte. Cliquer à l'extérieur de celle-ci n'entraînera aucune sorte d'action.

Les ***boîtes de dialogue*** à offrir forment, comme nous l'avons indiqué, des objets interactifs d'affichage destinés à apporter un retour d'informations précis à propos d'un événement qui vient de survenir suite à une action entreprise par l'utilisateur.

Beaucoup de personnes handicapées posent des actions sans trop savoir pourquoi. Ce n'est qu'après qu'elles réfléchissent aux conséquences de leurs actions. Ce type d'objet permettra donc au concepteur de l'application d'offrir à l'utilisateur le recul nécessaire à l'analyse de la validité du traitement qu'il a effectué. L'utilisation de la boîte de dialogue est donc un compromis entre le danger de distraire, de démotiver l'utilisateur et le désir de l'aider à la bonne exécution de la tâche.

Il existe différents types de boîtes de dialogue : les boîtes de dialogue **de confirmation**, **d'information** et **de guidage**. Le lecteur intéressé trouvera la description de ces différentes boîtes dans le document de I. Provot et J. Vanderdonck [Prov, 91].

Les boîtes de dialogue que nous offrirons seront composées d'un libellé, d'un pictogramme et de un ou deux boutons sur lesquels l'utilisateur peut appuyer pour confirmer sa lecture du message ou répondre à une question.

Les boutons d'une boîte de dialogue peuvent être de quatre types différents : boutons "OK" ou "ANNULER" et boutons "OUI" ou "NON". Ces deux derniers boutons seront préférés lorsque les messages traiteront de questions ou de confirmations. En effet, "OUI" et "NON" donnent une meilleure représentation de la confirmation et du refus que "OK" ou "ANNULER". Les boutons "OUI" et "OK" seront de couleur verte pour leur associer une connotation positive alors que les boutons "NON" et "ANNULER" seront colorés en rouge pour leur associer une connotation négative.

Ajoutons encore, pour clôturer ce paragraphe que nous avons choisi d'illustrer les boutons par leur libellé et un pictogramme. Ceci afin de renforcer leur signification et pour permettre à tous les utilisateurs de les comprendre. Pour les personnes sachant lire, le fait de présenter du texte est une source de motivation. En effet, il leur procure le sentiment que nous leur offrons un outil à la hauteur de leurs capacités. Le gain pour les personnes ne sachant pas lire est évident. De plus, le fait de présenter du texte leur permettra de "reconnaître" le mot s'ils le voient régulièrement associé au pictogramme. Ils ne pourront cependant faire la distinction entre "OK" et "OUI" et entre "ANNULER" et "NON".

4.1.4.La boîte d'édition

Dans plusieurs applications réalisées précédemment, il est demandé à l'utilisateur d'introduire de l'information au moyen du clavier. Nous avons dès lors décidé d'offrir un objet interactif permettant ce type d'introduction.

Cependant, ce moyen d'interaction pose souvent des problèmes dans le chef de l'utilisateur. Tout d'abord, le fait de devoir taper [Return] pour marquer la fin de la saisie

n'est pas toujours évident pour l'utilisateur. Il attend donc, après avoir introduit l'information que "quelque chose se passe". Alors que du côté application, il y a attente que l'utilisateur valide au moyen de la touche [Return]. Pour éviter ce blocage, le concepteur choisit en général d'associer au champ de saisie, un bouton de validation.

La ***boîte d'édition*** est un objet permettant à l'utilisateur d'introduire et de manipuler des chaînes de caractères en utilisant le clavier. Il est constitué d'un libellé, d'une invitation à saisir et d'une zone rectangulaire dans laquelle le texte est introduit. Nous y adjoindrons un bouton graphique de validation. Ce dernier sera de couleur verte et reprendra un pictogramme déterminé ainsi que le libellé "OK".

Une représentation en a déjà été donnée dans l'introduction de ce chapitre.

Bien que nous ayons choisi d'offrir ce type d'objet en raison de son utilisation quasi générale dans les mémoires précédemment réalisés, nous ne pouvons que mettre en garde le concepteur sur les problèmes qu'il peut occasionner pour les personnes présentant un handicap important. Une telle population aura des difficultés à passer du clavier à la souris et vice versa. La nécessité de remplir le champ de saisie pourra être fort perturbante pour la personne présentant une déficience importante.

Dans ce cas, la boîte d'édition pourra être avantageusement remplacée par un dérouleur. Encore faut-il connaître un ensemble de valeurs parmi lesquelles l'utilisateur peut choisir.

Un autre objet de saisie destiné aux personnes ne pouvant utiliser correctement le clavier aurait pu être offert. C'est l'"écritoire". Cet objet, proposé par F. Cheron et O. Berger [Cher, 90] dans leur travail d'évaluation du logiciel compte, est composé d'un dessin à l'écran des 26 lettres de l'alphabet, du "é", du "è", du trait d'union, de la barre d'espacement et d'un bouton de validation. Il permet d'éviter l'utilisation du clavier. Nous avons choisi de ne pas l'offrir car il n'est nécessaire que pour les personnes présentant un handicap important. Il pourra cependant être construit par le concepteur si celui-ci le désire à l'aide d'objets icônes.

4.1.5.L'écran

L'organisation des informations sur les divers écrans peut constituer un attrait supplémentaire pour l'utilisateur. Le concepteur engagé dans la réalisation des écrans de son application veillera à respecter certains principes.

Il est, par exemple, recommandé d'effectuer des regroupements logiques d'informations et d'offrir un renforcement visuel des regroupements. Car les utilisateurs, s'ils présentent un faible niveau de connaissances ont besoin d'écrans simples et logiquement organisés. Si on respecte, en plus, le principe d'homogénéité de représentation dans la succession des différents écrans de l'application, la durée et la qualité du processus d'apprentissage s'en trouveront améliorés.

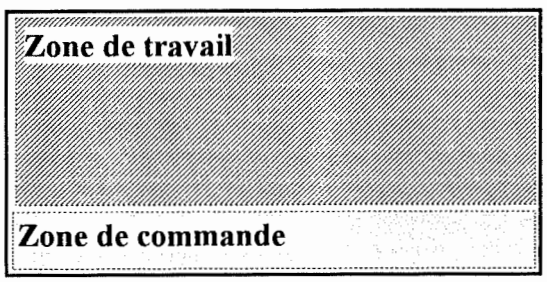
Nous avons par ailleurs pu observer, lors de la visualisation des logiciels réalisés dans le domaine du handicap, que les écrans étaient souvent constitués de deux zones.

Une zone regroupe les informations relatives au contenu de la tâche proprement dit. Nous l'appellerons "***zone de travail***". Cette zone reprend la partie de l'application permettant l'implication rationnelle et émotionnelle de la personne handicapée mentale dans le contexte de la tâche. C'est dans cette zone que "la métaphore du mini-monde" dont il est question dans le premier chapitre prend toute son importance. Elle est la partie motivante de l'application.

Une autre zone regroupe les informations relatives au déroulement de la tâche. Nous l'appellerons "***zone de commande***". Elle est constituée de boutons graphiques représentant des commandes affectant l'enchaînement des opérations.

En conséquence, l'objet ***écran*** offert sera décrit comme une zone rectangulaire de même taille que l'écran physique, zone dans laquelle la personne handicapée visualisera et modifiera directement les informations relatives à la tâche. Un écran est donc scindé en deux parties disjointes, une zone de travail et une zone de commande, dont le contenu et les opérations qui y sont associées sont déterminés par la tâche à réaliser.

Nous avons choisi d'offrir à l'utilisateur de nos outils la possibilité de définir un écran ne possédant pas de zone de commande ainsi que celle de définir un écran dont la zone de travail est illustrée par un pictogramme. La taille des zones ainsi que leurs couleurs seront définies par le concepteur. Il faut cependant signaler que la somme de la taille des zones est la taille de l'écran physique.



Remarque: L. Topet et X. Vonèche [Tope,91] distinguent trois zones logiques pour leurs écrans. La troisième zone appelée "***zone d'information***" ne nous a semblé utile que dans un ensemble restreint de situations. Elle est destinée principalement aux utilisateurs sachant lire et elle est purement statique. Nous avons donc décidé de ne pas l'offrir pour rester le plus général possible. Il est cependant possible au concepteur intéressé de la réaliser grâce à un objet icône comme nous le comprendrons aisément après lecture de la section suivante.

4.1.6.L'icône

Lors de l'analyse des mémoires précédemment réalisés, nous avons remarqué qu'un grand nombre des objets interactifs composant l'interface des applications entraient dans les classes d'objets définies précédemment.

Nous avons alors répertorié les objets restants et nous avons tenté de les classer. Pour ce faire, nous avons analysé leur comportement et nous avons pu déterminer un ensemble de propriétés guidant celui-ci.

Ces propriétés sont :

(In)sélectionnable : cette propriété indiquera si l'objet peut être sélectionné ou non par la personne handicapée. Un exemple d'objet sélectionnable offrant un feed-back informatif est le bouton illustré par un pictogramme représentant une oreille, du logiciel de connaissance du corps interne . "Cet objet permet de reproduire un message vocal digitalisé" [Font, 91].

(In)amovible : cette propriété signalera si l'objet peut être déplacé par la personne handicapée durant l'exécution du programme. La plupart des objets restants sont inamovibles. Un exemple d'objet amovible a été trouvé dans le logiciel d'apprentissage du corps humain [Lepo, 90], dans la partie de l'application qui permet de composer un puzzle d'un corps. Les différentes parties du corps peuvent être déplacées à travers l'écran. Lorsqu'elle clique sur l'objet, la personne handicapée voit l'objet "collé" au curseur de la souris. L'objet suivra celui-ci jusqu'à ce qu'elle clique à nouveau. Cela aura pour effet de "décoller" l'objet et de le positionner à l'écran à l'endroit du click.

(In)visible : cette propriété signalera si l'objet sera visible ou non durant le traitement. Ceci ouvre la possibilité de créer des boutons de commandes invisibles et pouvant être placés sur des pictogrammes ou des graphiques. Nous avons donc la

possibilité de sensibiliser certaines régions d'un dessin par simple adjonction d'un bouton de ce type. Dès ce moment, ces régions pourraient être initiatrices de diverses actions.

(Non)pile : cette propriété indique si l'objet peut être dupliqué par simple sélection lors de l'exécution du programme. Si un objet possède la propriété pile, il lui sera assigné en même temps la propriété sélectionnable. Lorsque l'utilisateur clique sur un objet ayant ces propriétés, un objet identique est créé. Cet objet est amovible mais n'est plus un objet "pile".

Défaut : dire qu'un objet interactif possède cette propriété revient à dire qu'il est sélectionnable, inamovible et visible.

A l'analyse des propriétés, nous remarquons que certaines d'entre elles sont incompatibles. En effet, il ne peut exister d'icône insélectionnable et amovible ou insélectionnable et pile. On peut également se poser la question de l'utilité d'un objet qui serait insélectionnable et invisible.

Nous avons donc décidé d'offrir un objet "à la carte". Cet objet, que nous appellerons ***icône*** sera illustré par un pictogramme. Son comportement sera déterminé par les propriétés que le concepteur lui aura attribuées. Outre celles-ci, le concepteur pourra déterminer la couleur de fond, la couleur du cadre entourant le pictogramme et la localisation de l'objet.

Nous aimerions par cet objet compléter l'ensemble des objets interactifs nécessaires au concepteur d'applications pour personnes handicapées. Ceci permettrait d'éviter d'avoir à définir des objets interactifs supplémentaires dans le contexte de la boîte à outils. Les personnes chargées de compléter cette dernière pourraient alors se pencher sur d'autres outils. Ces derniers seront décrits au chapitre 7. Nous sommes cependant conscients que nous ne pouvons prétendre y être parvenus tant qu'une évaluation des objets interactifs proposés n'aura pas été effectuée.

4.2. Analyse conceptuelle des objets graphiques

Nous allons, dans cette section, proposer l'analyse conceptuelle des objets graphiques selon le langage OBLOG. Nous commencerons par l'analyse des objets graphiques composants. Ces derniers étant les objets constitutifs de certains objets graphiques offerts, ils ne seront pas offerts tels qu'ils seront décrits. Nous les décrirons néanmoins par soucis de complétude. Quant aux objets graphiques offerts, ils seront décrits dans la seconde partie de cette section.

4.2.1. Les objets composants

Nous reprenons ici la description des objets graphiques composants. Ces objets sont : la flèche de défilement, le libellé, le pictogramme, le bouton de commande et l'élément de dérouleur.

A. La flèche de défilement

Définition : la flèche de défilement est un rectangle dans lequel est dessiné un triangle pointant dans la direction du défilement.

Représentation graphique :

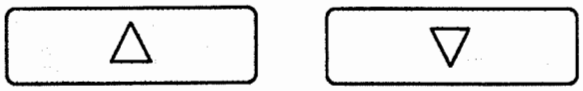
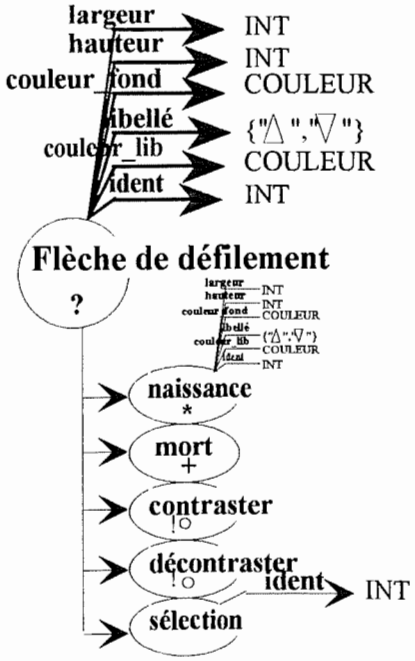


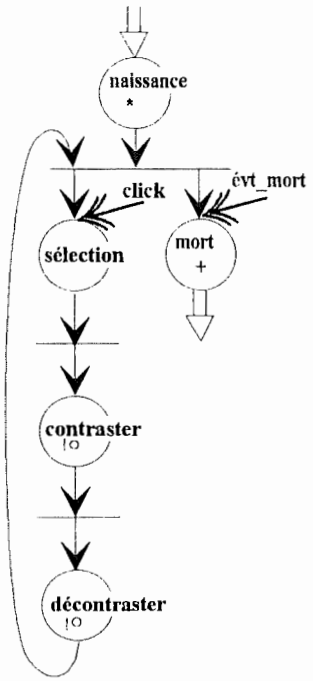
Diagramme matriciel :



Remarque: COULEUR est le type énuméré suivant :

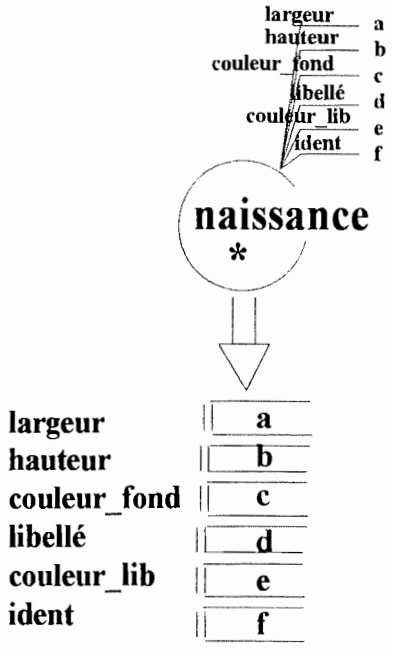
{blanc, noir, rouge, vert, jaune, bleu, cyan, magenta, brun, gris clair, gris foncé, rose, orange, bordeaux, kaki, azur}

Diagramme de comportement :



L'événement de mort sera déclenché par un événement externe.

Diagramme d'initialisation :



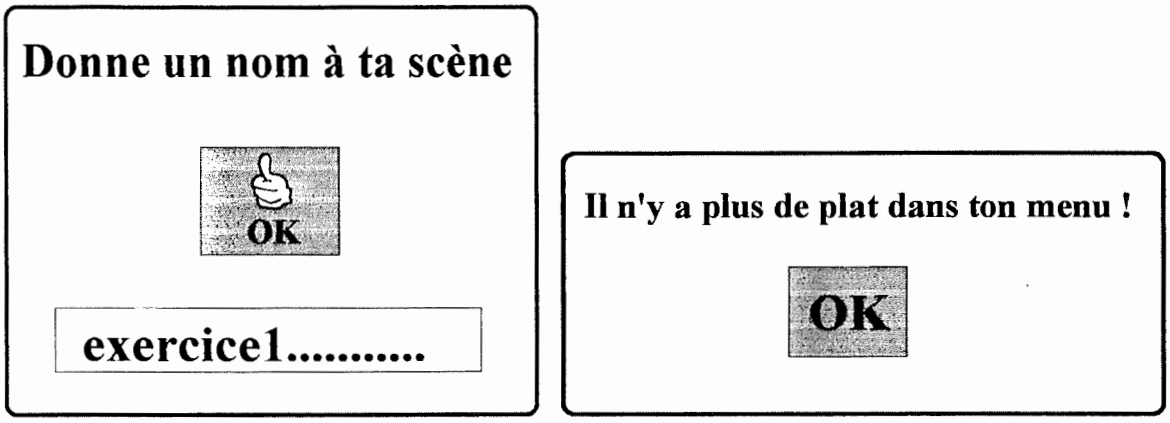
Pour créer une instance de flèche de défilement, il faut initialiser l'objet avec certaines valeurs. Ces valeurs lui seront passées par son événement de naissance qui sera appelé par un autre objet. Ce sera de cette manière que les objets composés instancieront leurs objets composants.

B. Le libellé

Définition : Le libellé est un intitulé textuel simple qui est utilisé pour définir le contenu informationnel d'un objet interactif ou en préciser la signification. C'est un objet graphique statique, il ne pourra y avoir aucune sorte d'interactions entre celui-ci et l'utilisateur.

Notez que les libellés peuvent être de divers types. Le libellé ***informatif***, le libellé **descriptif** et le libellé **identificatif** sont les trois types de libellés que nous avons rencontrés dans les mémoires précédents. Le lecteur intéressé par les divers types de libellé pourra en trouver une description dans le rapport de I. Provot et J. Vanderdonckt [Prov, 91].

Représentation graphique :



Dans le cas de ces deux boîtes de dialogue, les libellés sont le "OK" présent dans le bouton et les deux messages "Donne un nom à ta scène" et "Il n'y a plus de plat dans ton menu !".

Description en OBLOG : Comme nous l'avons mentionné précédemment, cet objet graphique possède un caractère purement statique. Nous ne le modéliserons pas comme un objet possédant une vie propre mais plutôt comme un type de donnée spécial car nous ne désirons pas alourdir inutilement la description des objets composés le contenant.

Sa représentation sera la suivante:

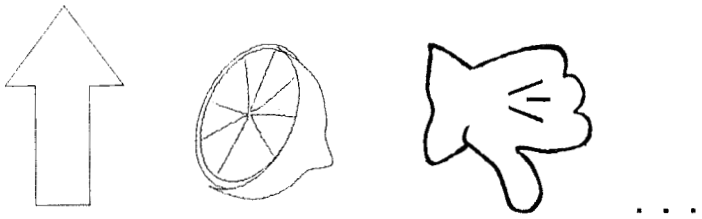
Libellé : CP[texte : STRING[1], couleur : COULEUR]

[1]Dans le cadre de cette modélisation, nous ne mentionnerons aucune limite pour la taille d'un string. Il est bien évident que cette taille sera limitée lors de l'implémentation. En OBLOG, STRING sera décrit comme une sequence de caractères (Seq[car]).

C. Le pictogramme

Définition : dessin positionné pour définir le contenu informationnel ou préciser la signification d'un objet interactif.

Représentation graphique :



Description en OBLOG : Pour les même raisons manifestées lors de la description du libellé, nous ne ferons pas de **pictogramme** un objet dans le sens OBLOG. Nous le décrirons plutôt comme un type de donnée.

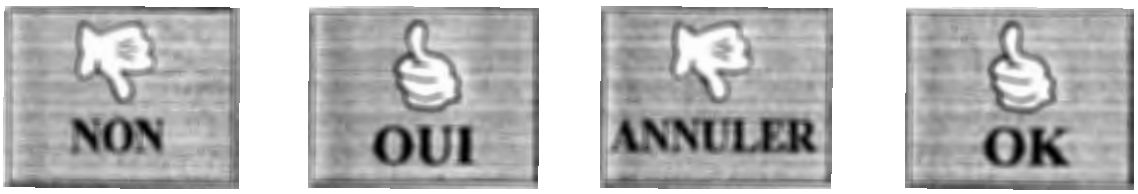
Sa description sera la suivante :

Pictogramme : CP[hauteur : int, largeur : int, dessin : DESSIN[2]]

D. Le bouton de commande

Définition : cet objet est exclusivement utilisé pour déclencher une action dont le résultat est décrit par un pictogramme et un libellé représentatif de l'action à déclencher. Un seul bouton peut être activé à la fois.

Représentation graphique :



Ces quatre boutons de commande seront des objets composants d'autres objets graphiques offerts.

---

[2]Un graphique pouvant être représenté de diverses manières, DESSIN est ici un type dont nous laisserons la détermination à l'implémentation.

Diagramme matriciel :

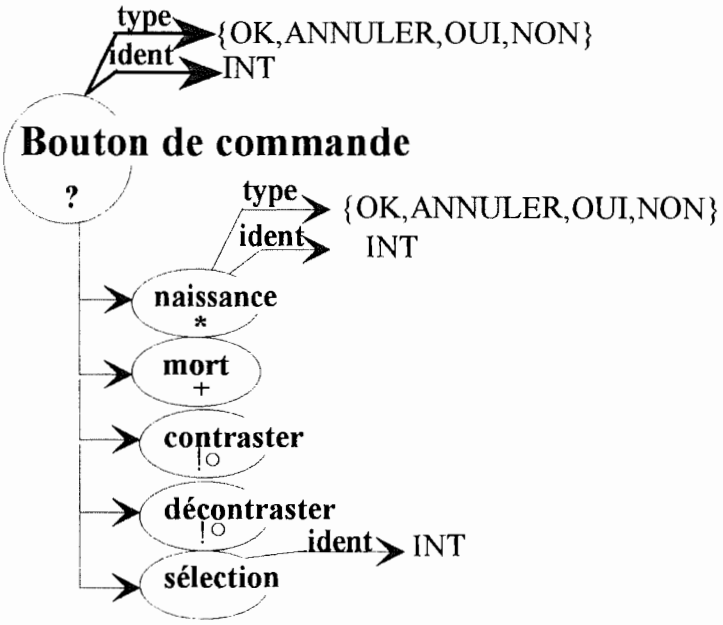


Diagramme de comportement :

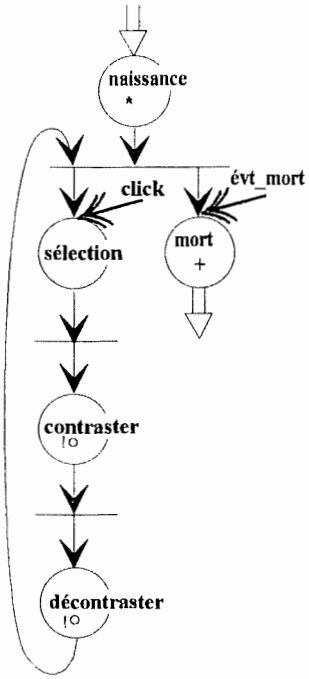
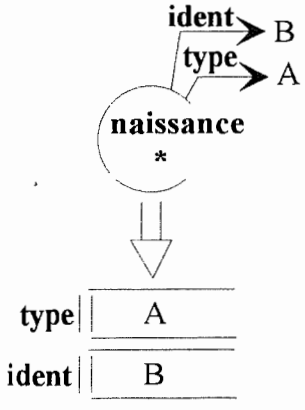


Diagramme d'initialisation :



E. l'élément de dérouleur

Définition : l'élément d'un dérouleur est une commande qui permet à la personne ayant un handicap de collaborer avec l'application par sa sélection. Dans le cadre de nos dérouleurs, cette sélection se fera soit par la sélection directe de l'élément au moyen de la souris, soit par le biais de la sélection d'une flèche de défilement qui entraînera la sélection d'un élément.

Comme nos dérouleurs peuvent être textuels ou graphiques, les éléments de ces derniers peuvent être constitués de libellés ou de pictogrammes.

Diagramme matriciel :

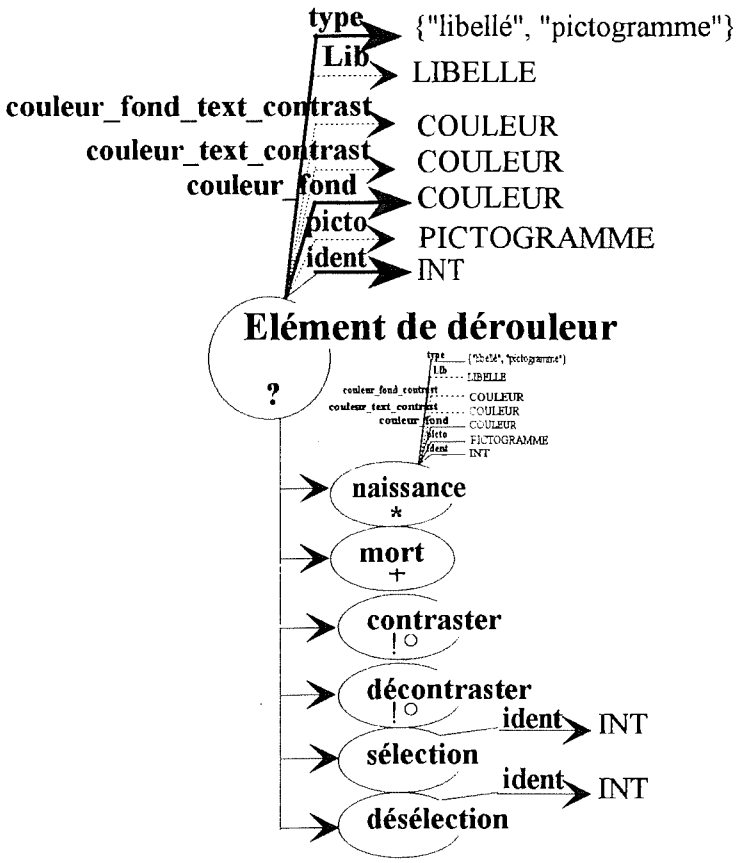
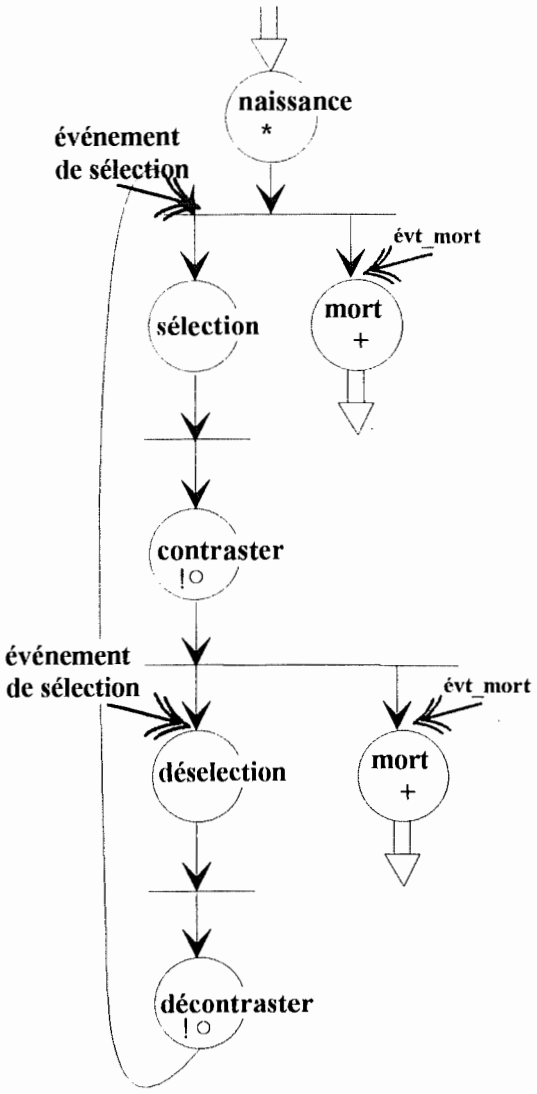


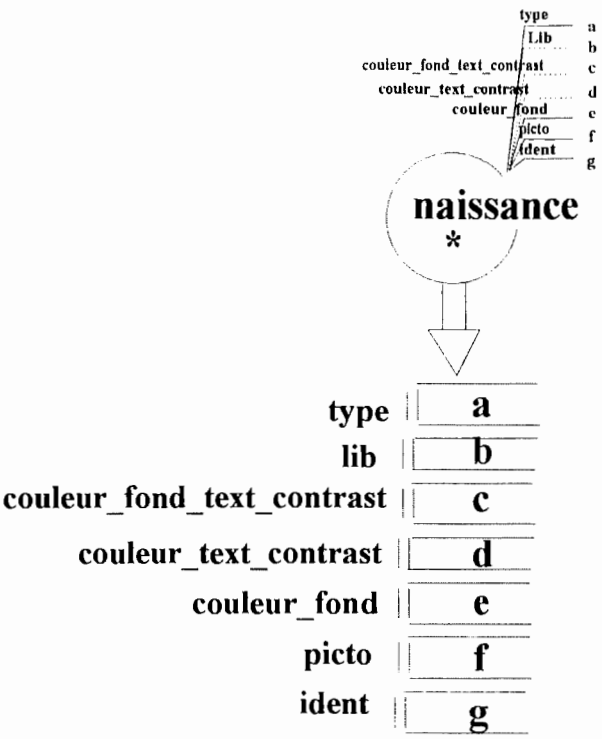
Diagramme de comportement :



Nous pouvons remarquer que c'est un événement de sélection qui entraînera le déclenchement de l'événement **sélection**. Nous avons donné ce nom générique à cet événement car, suivant le type du dérouleur, le type de sélection d'un élément de dérouleur sera différente. Il s'agira de la sélection directe de l'élément ou de la sélection par l'intermédiaire de la sélection d'une flèche de défilement.

Diagramme d'initialisation :

Dans le diagramme d'initialisation suivant, il faut remarquer que les attributs **couleur_fond_texte_contrast** et **couleur_text_contrast** n'auront une valeur que si l'attribut partiel **libellé** possède également une valeur. Dans ce cas, l'élément de dérouleur étant constitué d'un libellé, l'attribut partiel **picto** n'aura aucune valeur. Si l'attribut **picto** a une valeur, ce seront les trois attributs sus-mentionnés qui n'en auront plus.



4.2.2. Les objets graphiques d'interfaces pour personnes handicapées

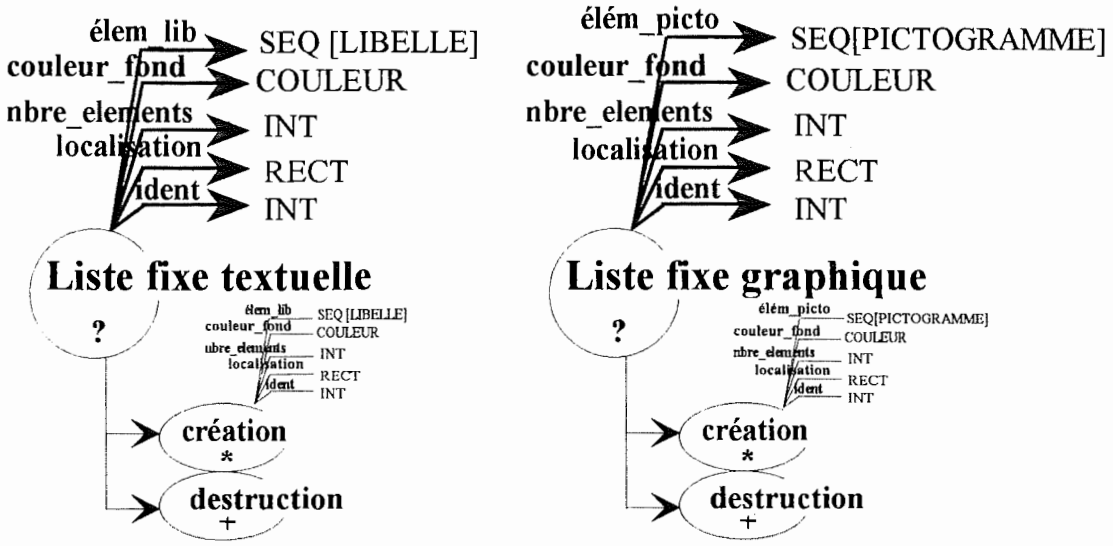
Ces objets seront offerts au programmeur pour qu'il puisse réaliser plus facilement les exercices destinés aux personnes handicapées. Les objets dont nous allons faire la description sont : la liste fixe textuelle et graphique, le dérouleur textuel et graphique à sélection par bouton, le dérouleur textuel et graphique à sélection directe, la boîte de dialogue, la boîte d'édition, l'écran et l'icône.

A. La liste fixe textuelle et graphique

Définition : Une liste fixe est un objet permettant d'afficher à l'écran une liste d'éléments. Ceux-ci pourront être soit des libellés, soit des pictogrammes. Cet objet n'a qu'une fonction informationnelle et aucun mécanisme de défilement ne lui sera associé. Il peut donc être considéré comme un objet statique

Représentation graphique : voir section 4.1.1

Diagramme matriciel :



Le type RECT est un type de donnée permettant à un attribut de représenter une zone rectangulaire à l'écran. Il est défini comme suit :

RECT : record [
POINT : ptsupgauche
POINT : ptinfdroit
]

Le type point permet à un attribut de représenter les coordonnées d'un point de l'écran. Il est défini comme suit :

POINT : record [
INT : x
INT : y
]

Diagramme de comportement :

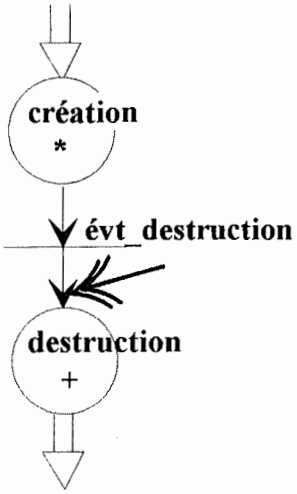
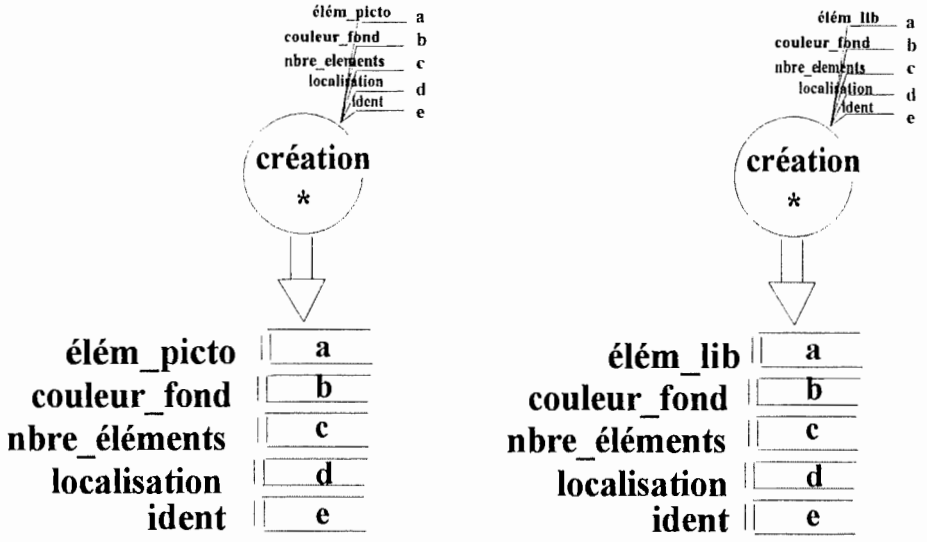


Diagramme d'initialisation :



Les diagrammes d'initialisation sont respectivement celui de la **liste fixe graphique** et celui de la **liste fixe textuelle.** Notez que **a** est une séquence de **c** éléments de type PICTOGRAMME ou LIBELLE.

B. Le dérouleur graphique et textuel à sélection par bouton

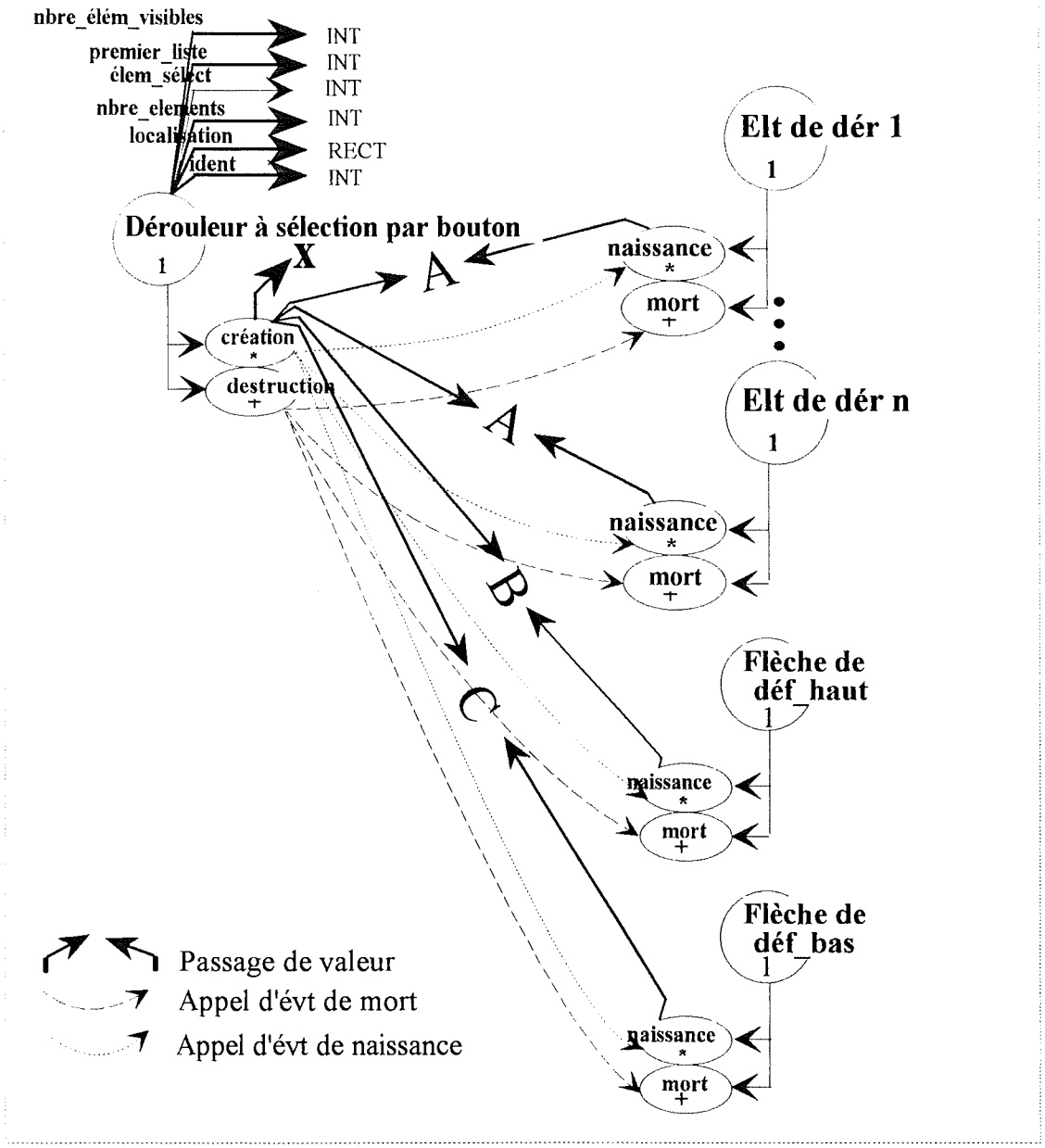
Définition : Cet objet contient une liste d'éléments déroulables et sélectionnables par click sur les flèches de défilement. Ces éléments pourront être soit des pictogrammes, soit des libellés.

Représentation graphique : Voir section 4.1.2.

Diagramme matriciel et d'interaction :

Ce type de diagramme n'est pas offert dans le formalisme OBLOG. Nous avons choisi de représenter le fait qu'un objet en "intègre" d'autres en reprenant le composé et les composants dans un cadre pointillé. Nous voulons indiquer par là que les divers objets composants repris dans le cadre sont indissociables de l'objet composé. De plus, nous y décrirons les diverses interactions entre objets.

Notez encore que nous avons dû séparer en divers cadres, la vie d'un objet pour des raisons de lisibilité. Les objets composants créés à la naissance de l'objet composé existent à tout moment de la vie de ce dernier même s'ils ne sont pas repris dans tous les cadres.

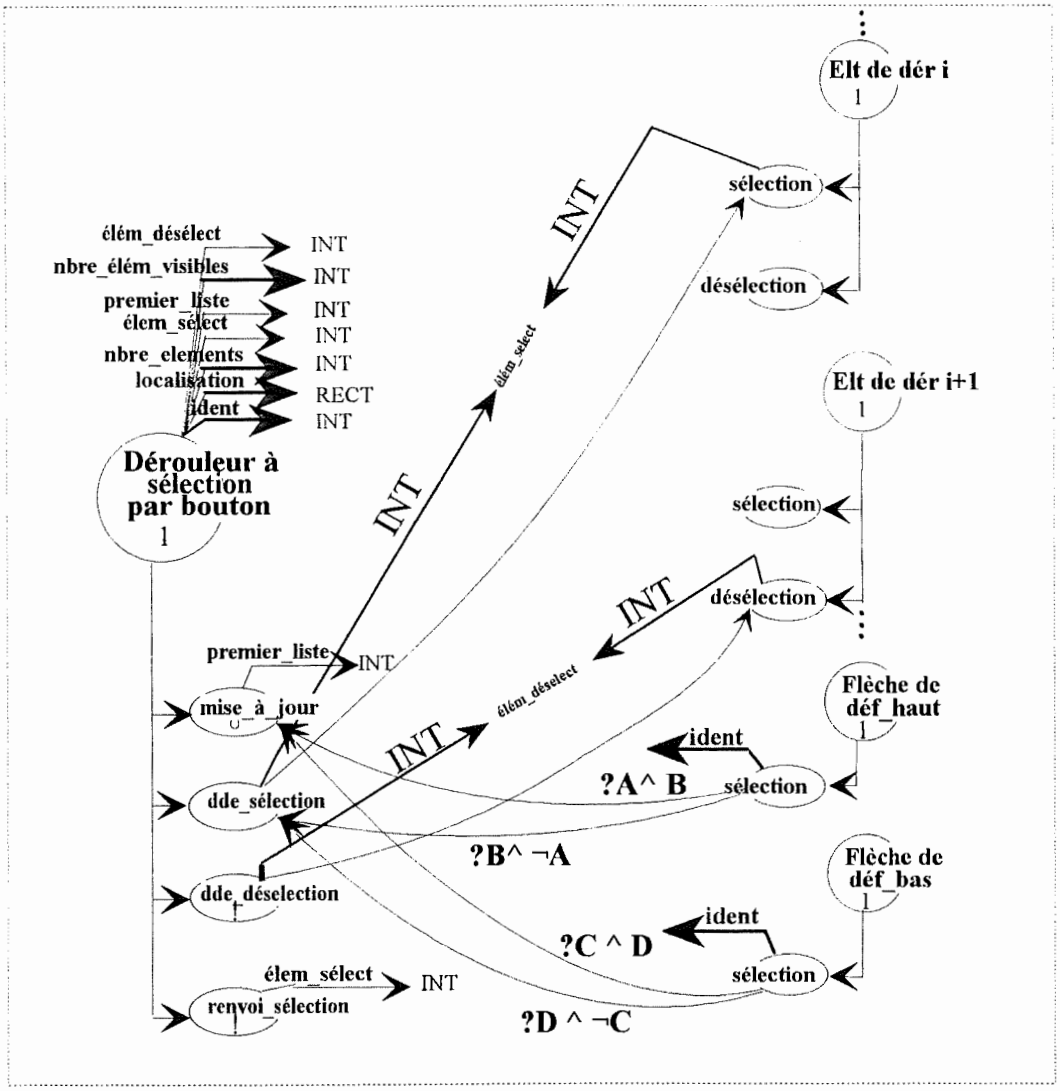


A : est l'ensemble des données qui seront passées à l'événement de naissance de l'objet **élément de dérouleur**. Ces attributs sont repris dans la description de l'événement de naissance d'élément de dérouleur. Notez que, les éléments de dérouleur pouvant être de type LIBELLE ou PICTOGRAMME, les attributs de l'événement de naissance correspondant à ce choix sont facultatifs.

B et C : sont l'ensemble des données qui seront passées respectivement à l'événement de naissance de l'objet **flèche de déf_haut** et à l'événement de naissance de l'objet **flèche de déf_bas**. Ces attributs sont repris dans la description de l'événement de naissance **de flèche de défilement**. Notez qu'à l'attribut **libellé** de flèche de déf_haut et flèche de déf_bas sera respectivement assignée la valeur " Δ " et la valeur " ∇ ". De plus, la

valeur de l'**identifiant** sera -1 pour la flèche de défilement vers le haut tandis que l'identifiant de l'autre flèche sera 1.

X : Toutes les valeurs qui serviront à initialiser les attributs de l'objet dérouleur ou qui permettront la création des objets composants.



?A : premier_liste > 1

?B : élém_sélect > 1

?C : (premier_liste + nbre_élém_visibles -1) <> nbre_éléments

?D : élém_sélect <> nbre_éléments

Pour ne pas alourdir inutilement le schéma, nous n'avons pas relié le transfert de l'indentifiant entre l'événement **sélection** des flèches de défilement et les événements **mise_à_jour** et **dde_sélection.** Ce transfert est cependant effectif.

Diagramme de comportement :

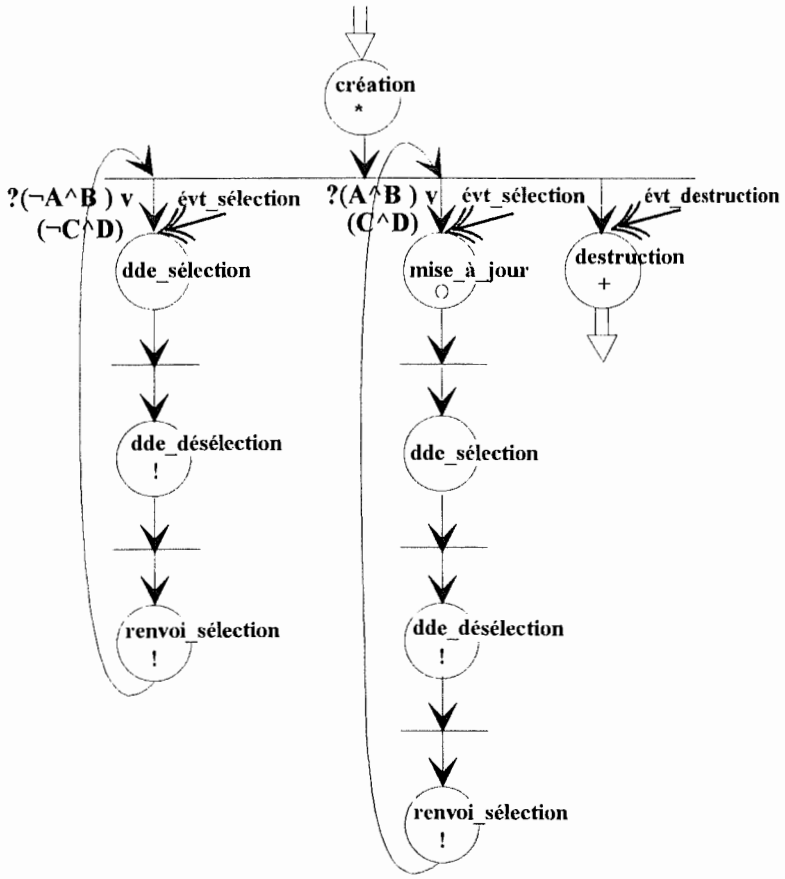
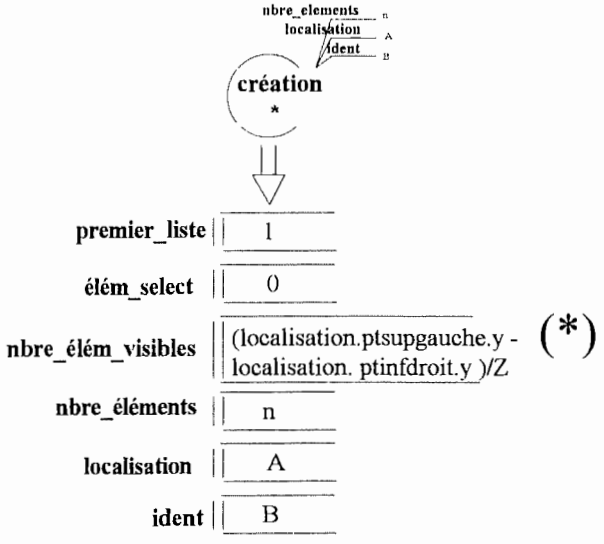
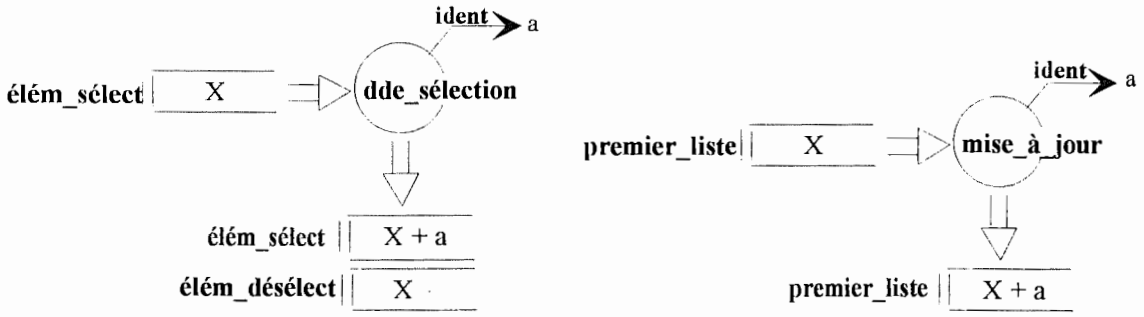


Diagramme d'initialisation :



(*) Z est la hauteur d'un élément de dérouleur. Lors de l'implémentation, elle sera fixe pour un élément textuel car nous n'utiliserons que des polices de caractères standards. Pour un élément de dérouleur graphique, elle sera de la taille du dessin correspondant.

Diagramme de mise à jour :



La valeur de l'identifiant passée à ces événements de mise à jour correspond aux identifiants des deux flèches de défilement. Comme nous leur avons assigné les valeurs 1 et -1, respectivement pour le défilement vers le bas et vers le haut, nous pouvons incrémenter ou décrémenter les valeurs de **premier_liste** et **élém_sélect** en modélisant un seul événement pour chaque assignation. L'autre solution, plus complexe aurait été de créer deux événements pour chacune des deux assignations, un pour une sélection ou un défilement vers le haut et un autre pour une sélection ou un défilement vers le bas.

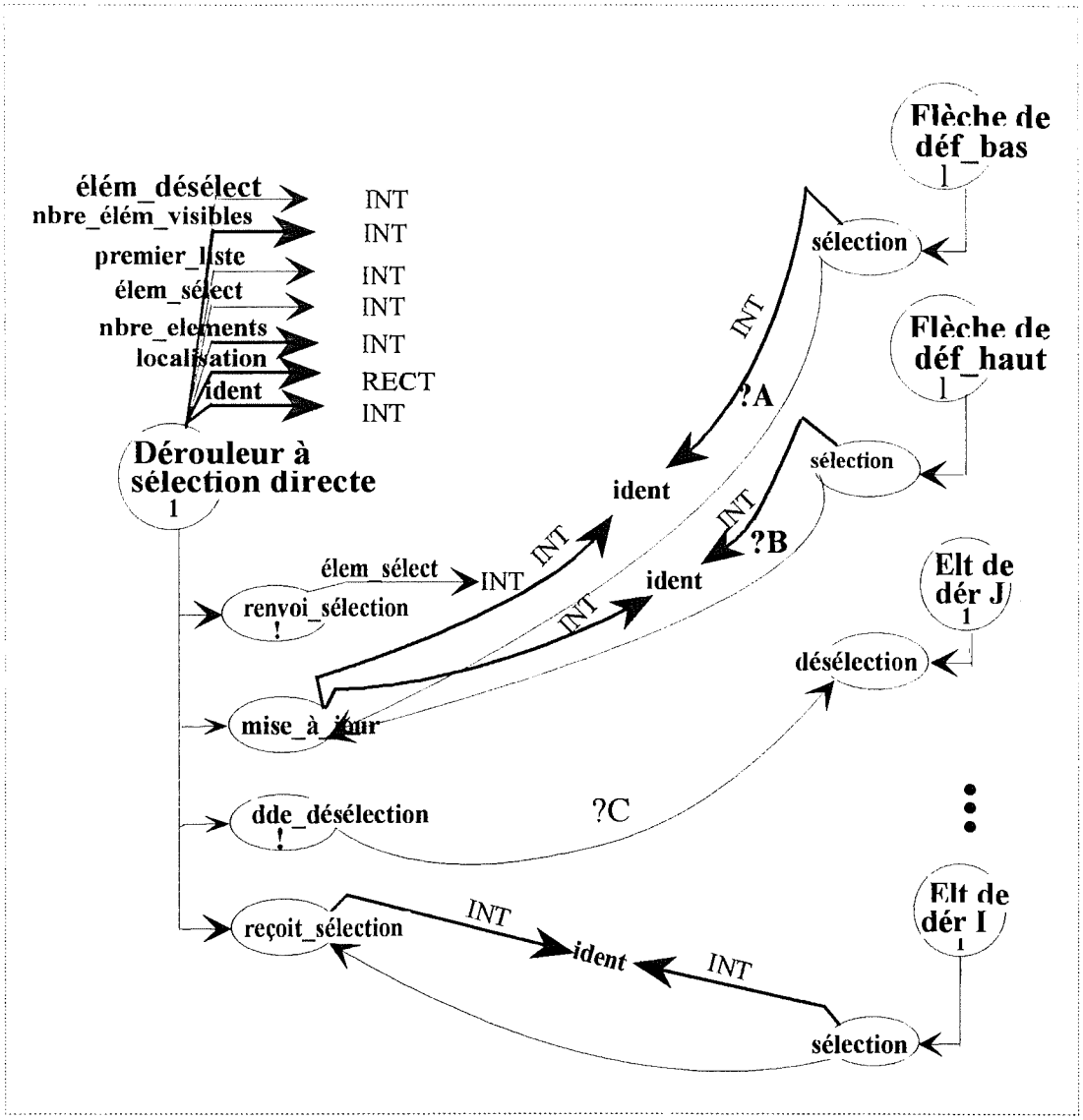
C. Dérouleur textuel et graphique à sélection directe

Définition : Cet objet contient une liste d'éléments déroulables par click sur les flèches de défilement et sélectionnables par click sur les éléments. Ces éléments pourront être soit des pictogrammes, soit des libellés.

Représentation graphique : Voir section 4.1.2.

Diagramme matriciel et d'interaction :

L'encadré des **créations** et **destructions** et les remarques correspondantes sont identiques à ceux du dérouleur à sélection par bouton.

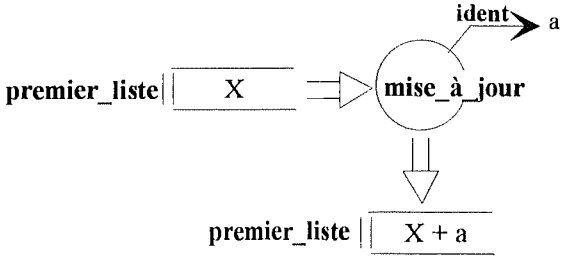


?A : premier_liste < nbre_éléments - nbre_élém_visibles + 1

? B : premier_liste > 1

?C : ident elt de der (J) = élem_désélect

Diagramme de mise à jour :



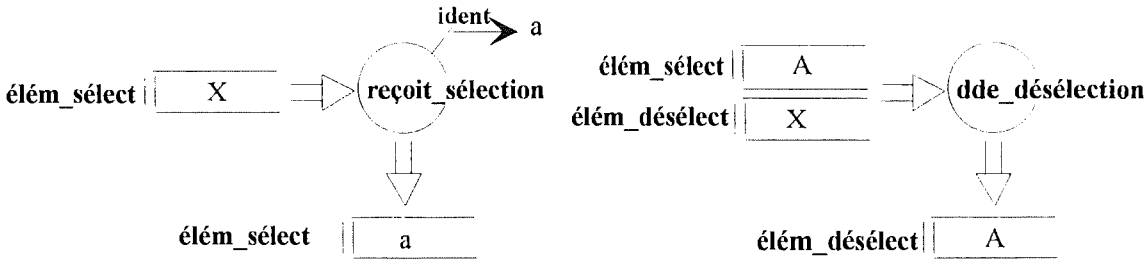


Diagramme de comportement :

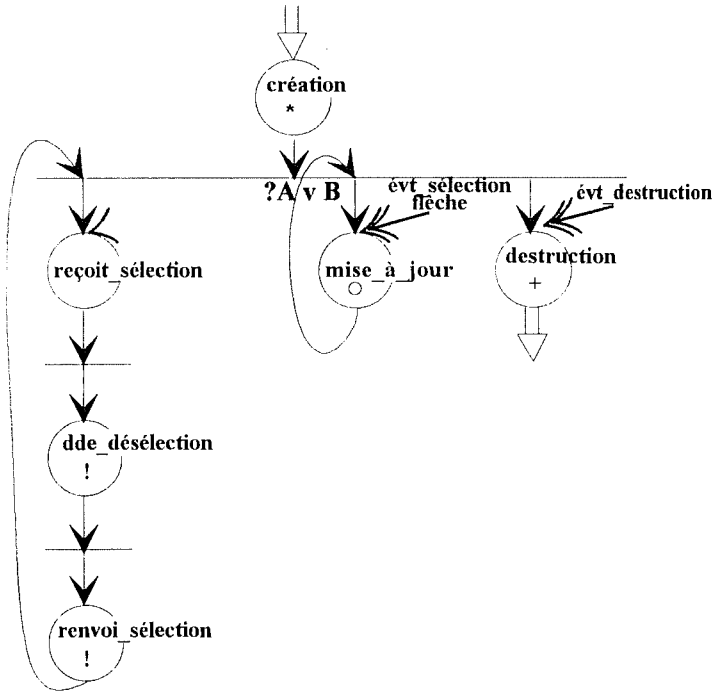
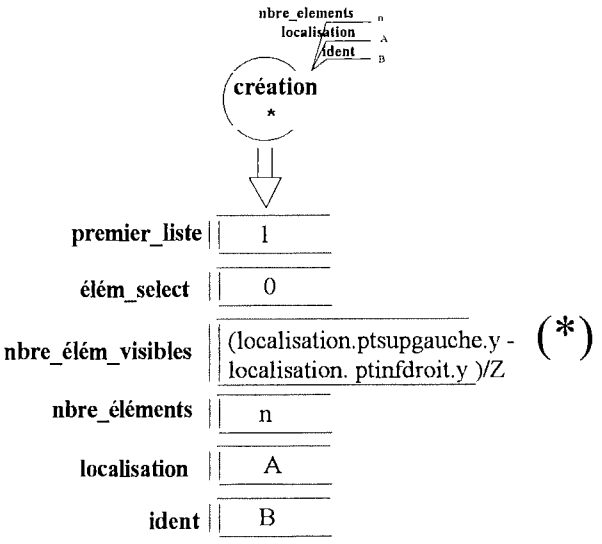


Diagramme d'initialisation :

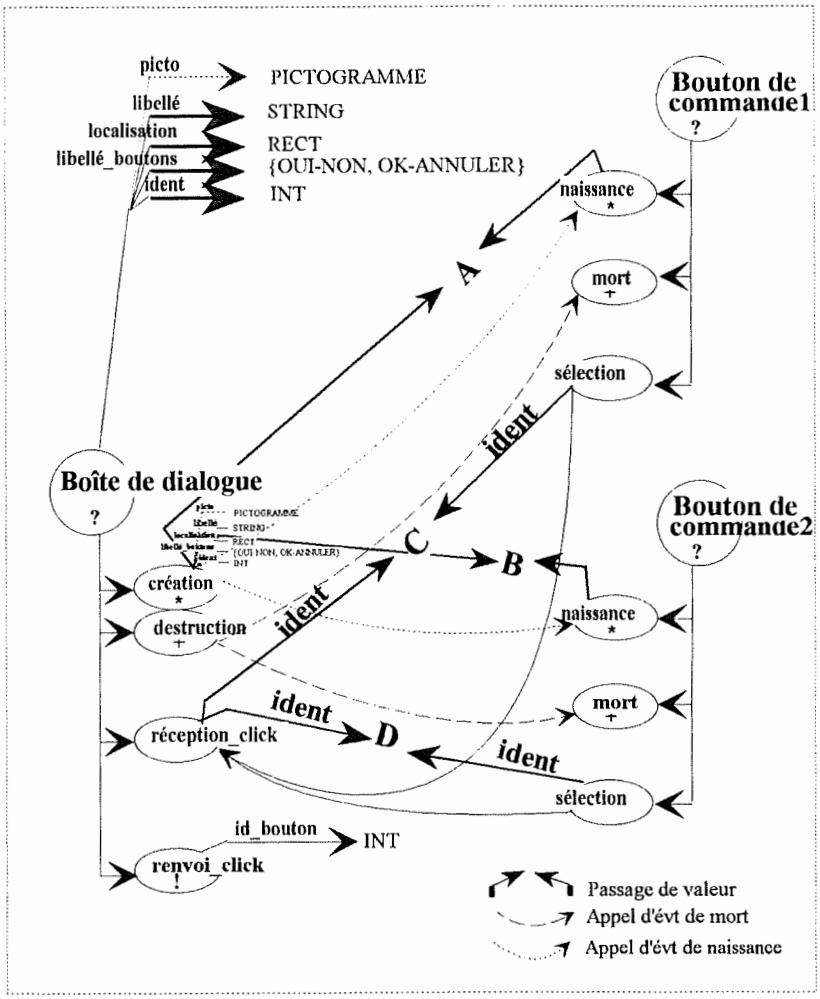


D. La boîte de dialogue

Définition : ce type d'objet graphique sert à apporter un feed-back à un événement s'étant produit suite à une action effectuée par l'utilisateur. Les boîtes de dialogue reprises dans les différents logiciels visionnés sont exclusivement modales. Elles sont composées d'un libellé et de un ou deux bouton(s) de commande. Elles sont aussi parfois illustrées par un pictogramme. Nous donnerons la description des boites de dialogue à deux boutons. La description des boites de dialogue à un seul bouton peut en être aisément déduite.

Représentation graphique : Voir section 4.1.3.

Diagramme matriciel et d'interaction :



A et **B** : représentent les deux ensembles de valeurs permettant de créer chacun une instance de la classe d'objets **bouton_de_commande**. Notez toutefois que si le libellé du **bouton de commande1** est "OK", le **bouton de commande2** ne pourra avoir que "ANNULER" comme libellé. De la même façon, les libellés "OUI" et "NON" sont associés.

Diagramme de comportement :

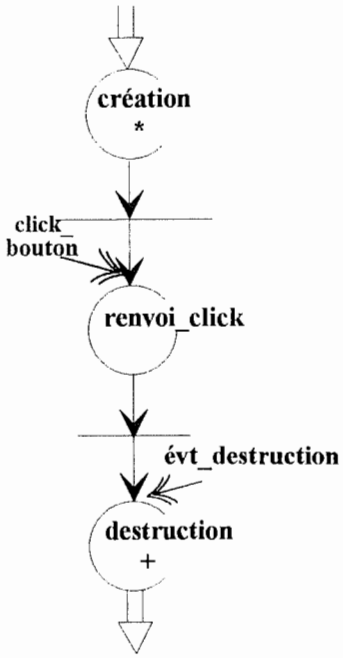
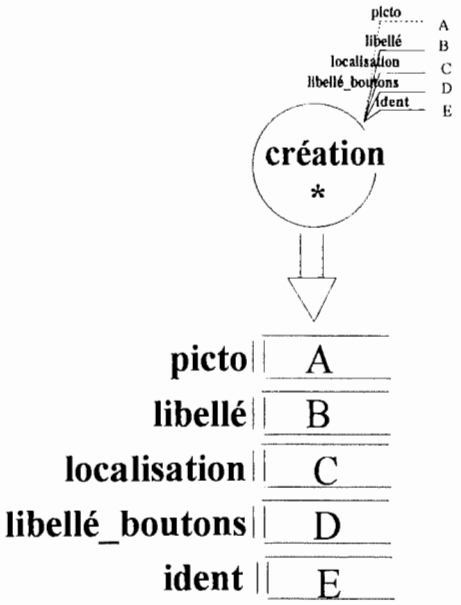


Diagramme d'initialisation :



E. La boîte d'édition

Définition : Zone de dialogue permettant à l'utilisateur d'introduire et de manipuler une chaîne de caractères par l'utilisation du clavier.

Représentation graphique :

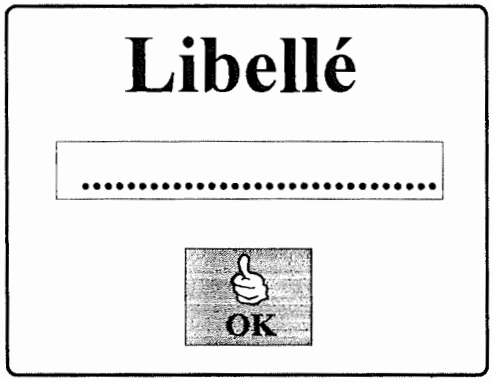
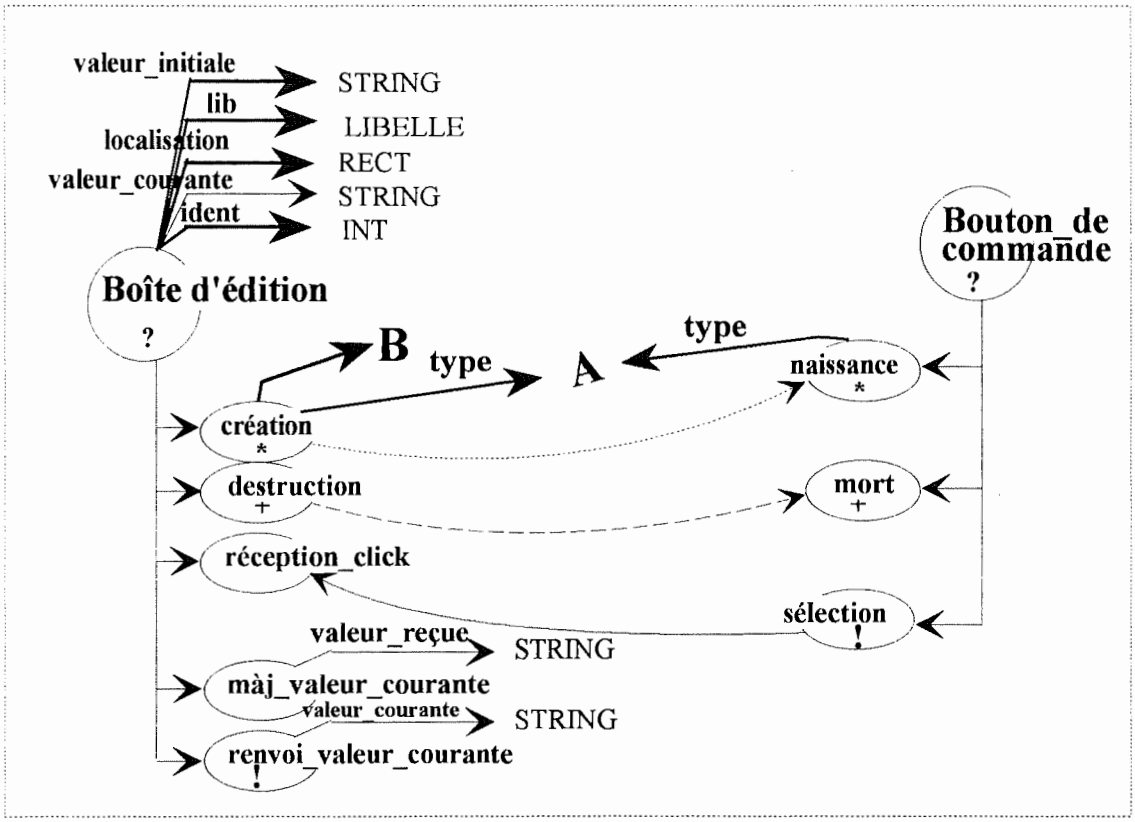


Diagramme matriciel et d'interaction :



B : représente l'ensemble des valeurs permettant de créer une boîte d'édition et d'initialiser l'attribut **type** du bouton de commande de la boîte.

L'événement **màj_valeur_courante** sera déclenché par le système chaque fois que la personne handicapée effectuera une action d'édition. Le système communiquera en outre à l'événement, la valeur qui est contenue dans la zone d'édition.

Diagramme de comportement :

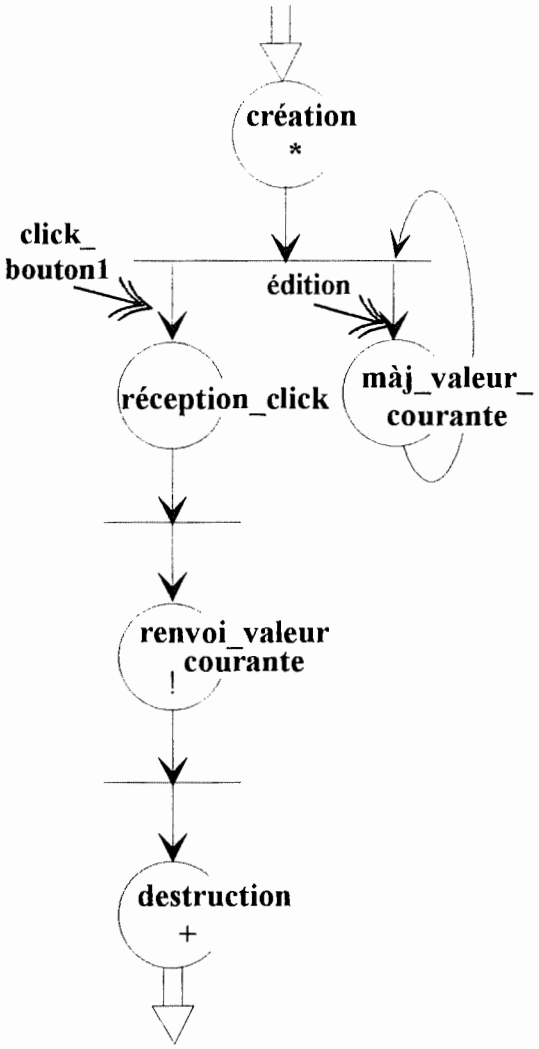


Diagramme d'initialisation :

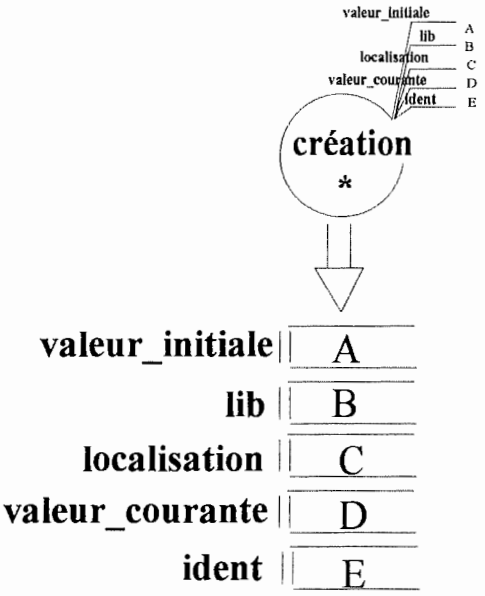
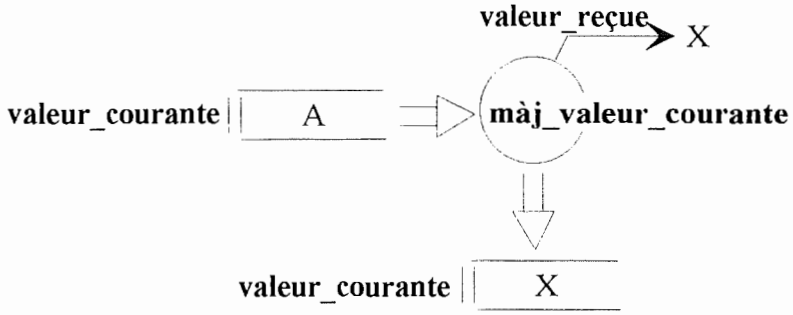


Diagramme de mise à jour :



F. L'écran

Définition : Zone rectangulaire de même taille que l'écran physique qui contiendra les divers objets graphiques constituant l'application.

Représentation graphique : voir section 4.1.5.

Diagramme matriciel :

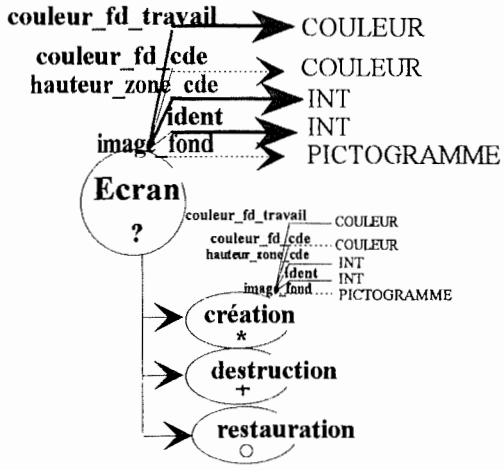


Diagramme d'initialisation :

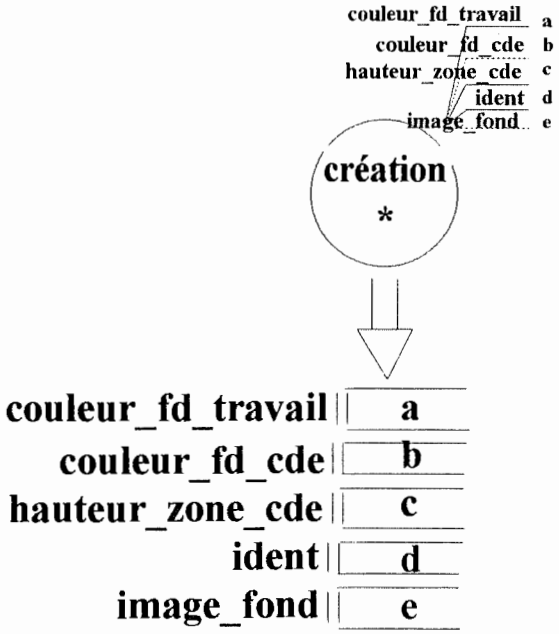
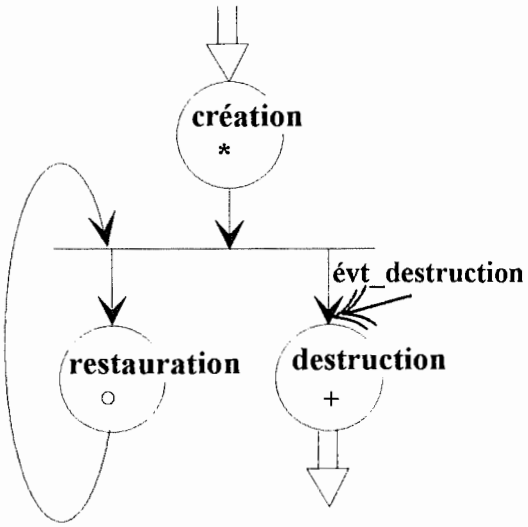


Diagramme de comportement :

L'événement **restauration** sera déclenché par le "système" chaque fois qu'un objet graphique contenu dans l'écran disparaîtra de ce dernier. La restauration sera d'ailleurs effectuée par le système(événement externe).



G. L'icône

Définition : Cet objet est illustré par un pictogramme qui peut représenter un type d'information, un état du système, une option, un paramètre, une action, etc. Il peut avoir différents comportements. Ces derniers sont décrits dans la section 4.1.6..

Diagramme matriciel :

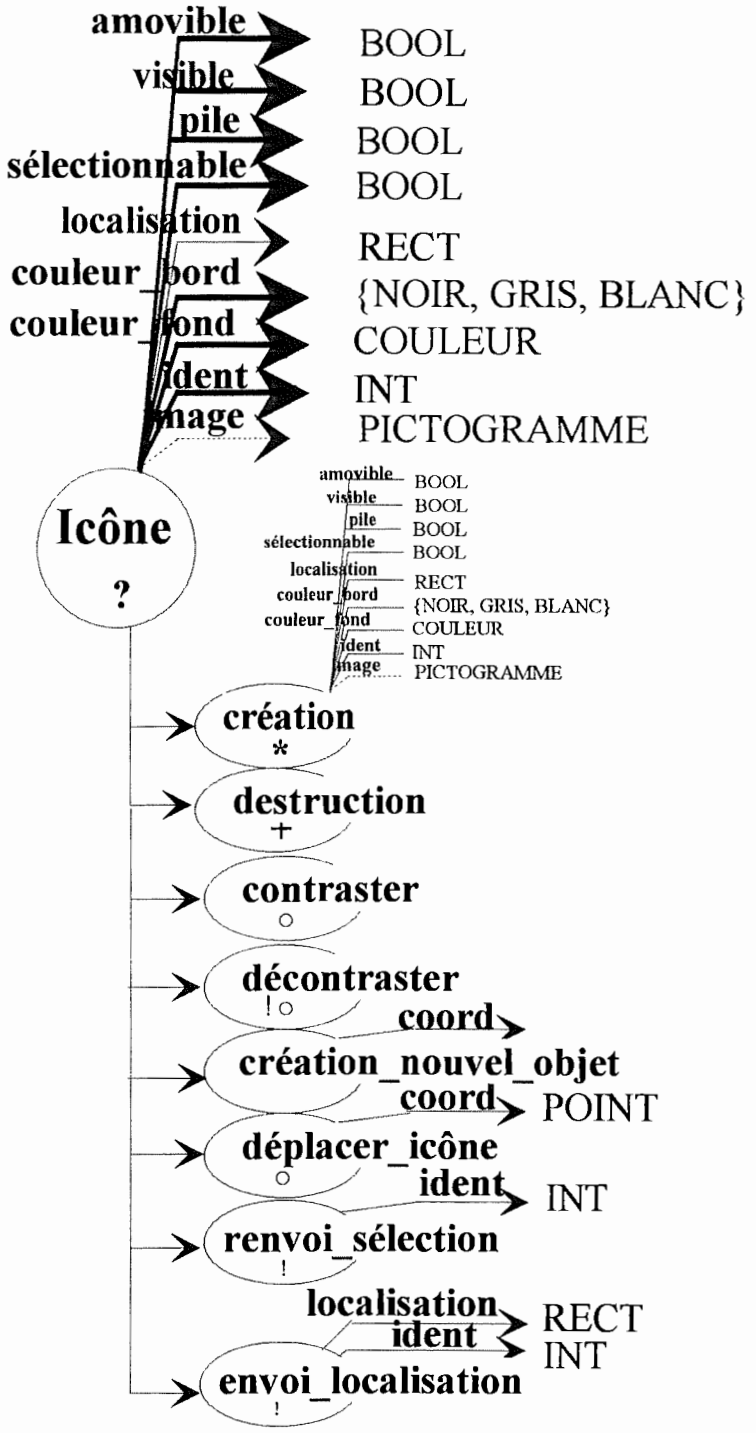


Diagramme de comportement :

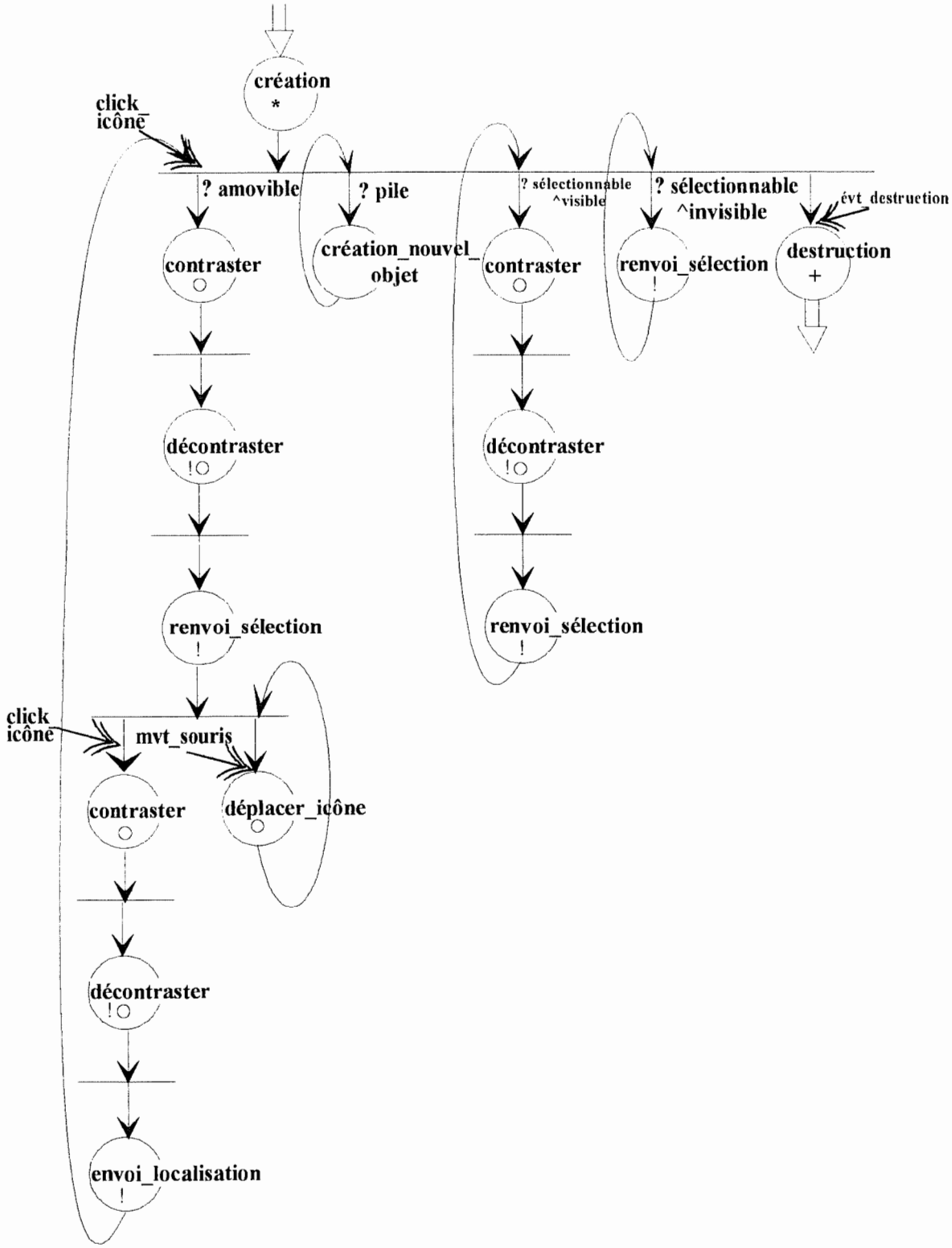


Diagramme d'initialisation :

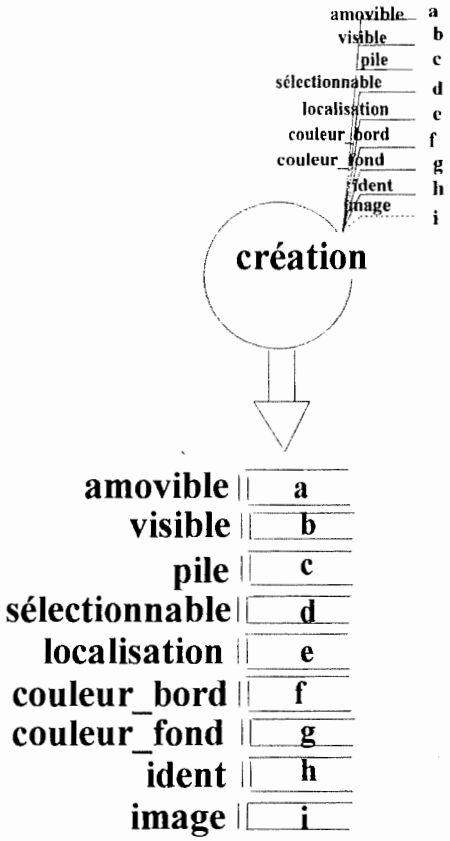
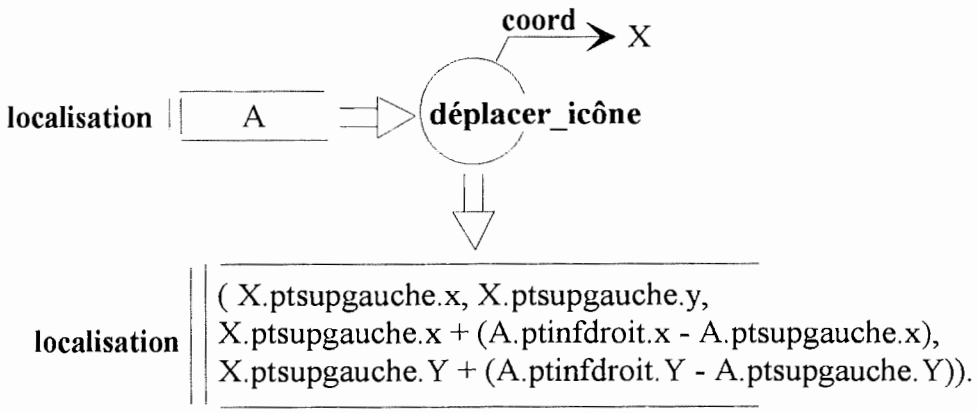


Diagramme de mise à jour :



Cette mise à jour permet de faire en sorte que l'attribut **localisation** contienne à tout moment la localisation courante de l'objet.

Chapitre 5 : Spécification d'exercices sur base de nos objets graphiques

Introduction

Nous allons tenter, dans ce chapitre, de montrer la pertinence des choix qui furent effectués lors de la détermination des outils et la multitude des traitements réalisables grâce à eux.

Pour ce faire, nous proposons la description des scénarios et la modélisation de deux exercices du cahier des charges du "Trèfle" et de deux parties d'interfaces provenant des mémoires réalisés précédemment dans le domaine du handicap. Les exercices implémentés seront visualisés lors de la défense du travail.

5.1. Exercice de désignation des membres du corps humain

L'exercice de désignation des membres du corps humain est tiré du mémoire de Thierry Lepoutre et de Jean-Michel Roquet [Lepo, 90]. Nous l'avons choisi car il nous permet de montrer une partie des traitements que l'on peut effectuer grâce à l'objet icône.

5.1.1. Scénario de l'exercice

Le scénario, que nous allons établir ici, ne correspond pas exactement au scénario établi par Roquet et Lepoutre. Dans leur mémoire, ces derniers utilisent divers paramètres. L'exercice se comporte différemment selon la valeur de ces derniers. Citons par exemple, les différents modes dans lesquels un exercice peut être effectué. Il peut être en mode passif et à ce moment, l'utilisateur ne fait que contempler les éléments du corps qui se désignent les uns après les autres. Il peut également se trouver en mode évaluation ou en mode apprentissage. Les utilisateurs doivent alors désigner eux-mêmes les divers éléments qui leur sont demandés.

Bien que ces traitements puissent être réalisés complètement, nous avons légèrement simplifié l'exercice afin d'alléger les spécifications en OBLOG et de gagner en compréhension.

L'exercice consiste donc à désigner, selon la demande, une partie du corps humain. Si l'utilisateur désigne correctement à l'aide de la souris la partie qui lui est demandée, il obtient un renforcement positif. Dans le cas contraire, rien ne se produit. S'il veut finir l'exercice, il clique sur le bouton de commande "FINI" et une boîte de dialogue apparaît lui demandant confirmation. S'il veut stopper l'exercice, le bouton "STOP" lui permet de faire apparaître une autre boîte de confirmation. Notons toutefois que la différence entre "FINI" et "STOP", à savoir que "STOP" permet de mémoriser la situation afin de reprendre l'exercice à l'endroit où l'utilisateur se trouvait, ne sera pas modélisée. Nous pouvons examiner l'interface de l'exercice à la figure 5.1.

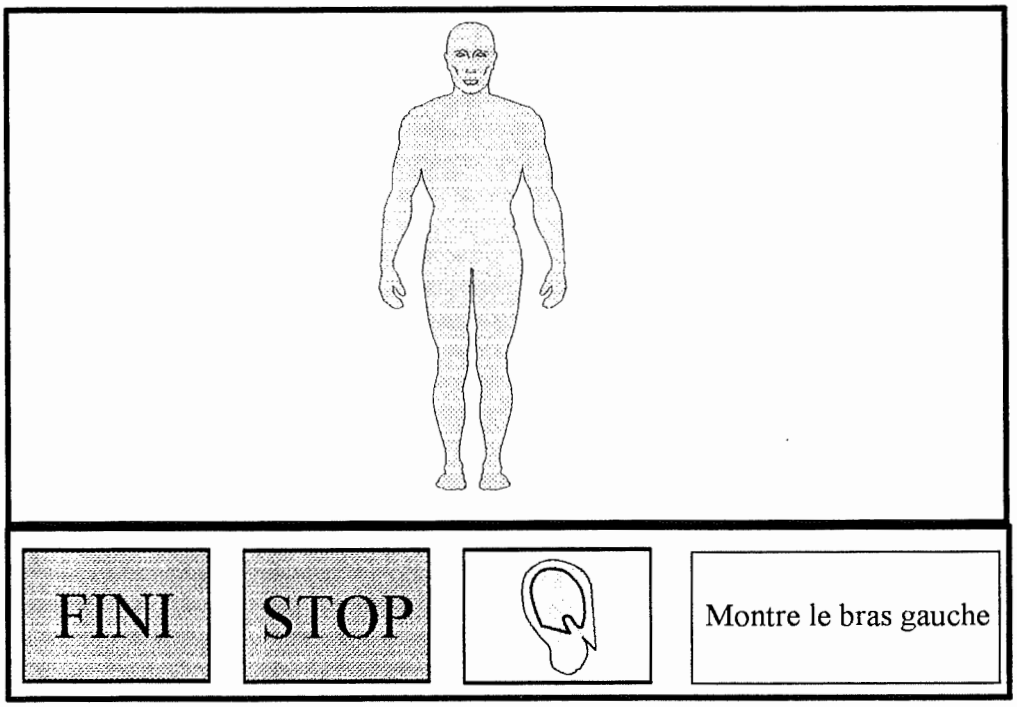


Figure 5.1. : interface de l'exercice de désignation du corps humain

5.1.2. Modélisation de l'exercice

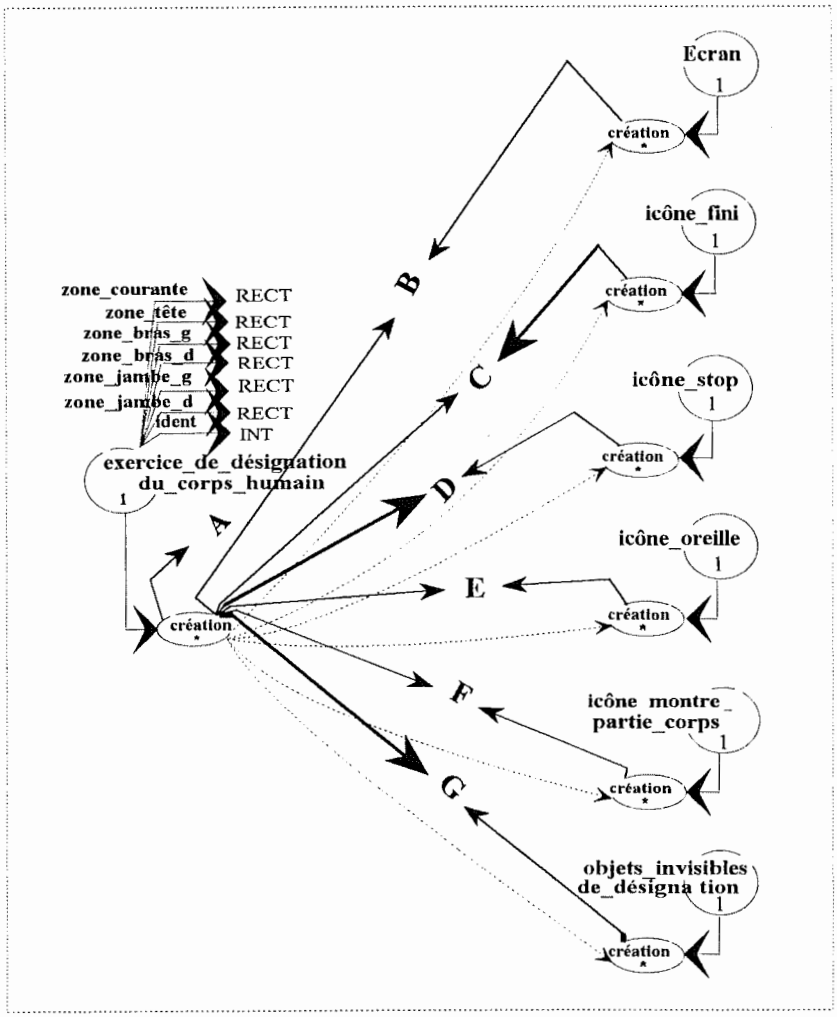
Pour réaliser l'exercice, nous aurons besoin de l'objet écran que nous créerons avec un pictogramme représentant un corps humain complet. Nous allons définir une zone de commande qui contiendra quatre objets icônes. Les trois premiers seront sélectionnables et représenterons des boutons de commande. Il s'agit des boutons "STOP", "FINI" et du bouton illustré par une oreille. La sélection de ce dernier n'entraînera aucune action dans notre exercice. Dans l'exercice original, la sélection de ce bouton entraînait la prononciation de la partie du corps à désigner. Le quatrième représentera un message indiquant quels sont les objets à sélectionner. Pour désigner les différentes parties du corps,

nous allons avoir recours à des objets icônes invisibles et sélectionnables que nous superposerons au fur et à mesure de l'avancement de l'exercice avec les diverses parties du corps. Suite à la sélection correcte d'une icône invisible, nous la détruirons et en créerons une autre sur une autre partie du corps. De cette façon, un et un seul membre peut être sélectionné à la fois; celui requis par le message se trouvant dans la zone de commande

Notons encore que, pour ne pas introduire trop de schémas, nous avons choisi de donner les initialisations d'attributs sous forme de texte. Le lecteur désireux de voir les graphiques correspondants les trouvera, sous forme générale au chapitre 4.

Diagramme matriciel et d'interaction :

Comme pour la description des objets graphiques dans le chapitre précédent, l'objet et ses composants seront entourés d'un rectangle en pointillé. Cela signifie que les objets composants sont inclus dans l'objet composé et que nous décrivons les interactions entre les composants et le composé dans ce même diagramme.



A : constitue l'ensemble des données nécessaires pour créer l'exercice. Cela comporte les attributs de l'objet **Exercice_de_désignation_du_corps_humain**, mais également les attributs nécessaires à la création des divers objets composants.

Les initialisations des attributs de l'exercice seront effectuées comme suit :

zone_tête = ((264,22),(388,146)),
zone_bras_g = ((363,125),(441,146)),
zone_bras_d = ((168,125),(246,146)),
zone_jambe_g = ((339,255),(375,333)),
zone_jambe_d = ((249,255),(285,333)),
ident = 1.

B : constitue l'ensemble des attributs nécessaires à la création de l'objet **Ecran.** Les valeurs passées seront :

ident = 1
hauteur_zone_cde = 90
couleur_fd_cde = blanc
couleur_fd_travail = blanc
image_fond = "personnage"

C : constitue l'ensemble des attributs nécessaires à la création de l'objet **icône_fini.** Les valeurs passées seront :

ident = 100
image = "picto_fini"
couleur_fond = vert
couleur_bord = noir
localisation = ((20,10),(84,74))[1]
visible = vrai
amovible = faux
pile = faux
sélectionnable = vrai

D : constitue l'ensemble des attributs nécessaires à la création de l'objet **icône_stop.** Les valeurs passées seront identiques aux précédentes sauf pour :

[1] Nous pouvons remarquer ici que ces coordonnées sont en rapport avec le coin supérieur gauche de la zone de commande. Ce dernier étant (0,0).

ident = 101
image = "picto_stop"
couleur_fond = rouge
localisation = ((188,10),(252,74))

E : constitue l'ensemble des attributs nécessaires à la création de l'objet **icône_oreille.** Les valeurs passées seront identiques aux précédentes sauf pour :

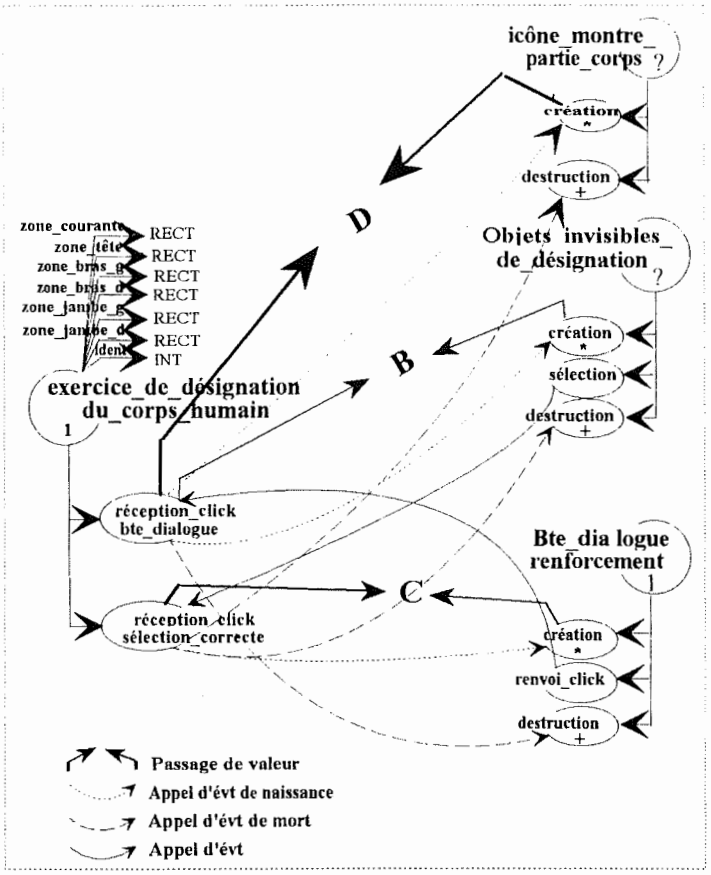
ident = 102
image = "picto_oreille"
couleur_fond = brun
localisation = ((84,10),(148,74))

F : constitue l'ensemble des attributs nécessaires à la création de l'objet **icône_montre_partie_corps.** Les valeurs passées seront identiques aux précédentes sauf pour :

ident = 103
image = "picto_montre_b_g"
couleur_fond = blanc
couleur_bord = blanc
localisation = ((272,10),(460,74))
sélectionnable = faux

G : constitue l'ensemble des attributs nécessaires à la création de l'objet **icône_invisible_de_désignation.** Les valeurs passées seront identiques aux précédentes sauf pour :

ident = 104
image = NULL
couleur_fond = blanc
couleur_bord = blanc
localisation = ((363,125),(441,146))
sélectionnable = vrai
visible = faux



B : constitue l'ensemble des attributs nécessaires à la création de l'objet **objets_invisibles_de_désignation.** Les valeurs passées seront identiques à celles passées lorsque l'on crée l'objet **objets_invisible_de_désignation** lors de la création de l'exercice sauf pour l'attribut **localisation**. Ce dernier prendra la valeur de la zone courante. Cette zone est mise à jour par l'événement **réception_click_sélection_correcte** comme indiqué après le diagramme de comportement.

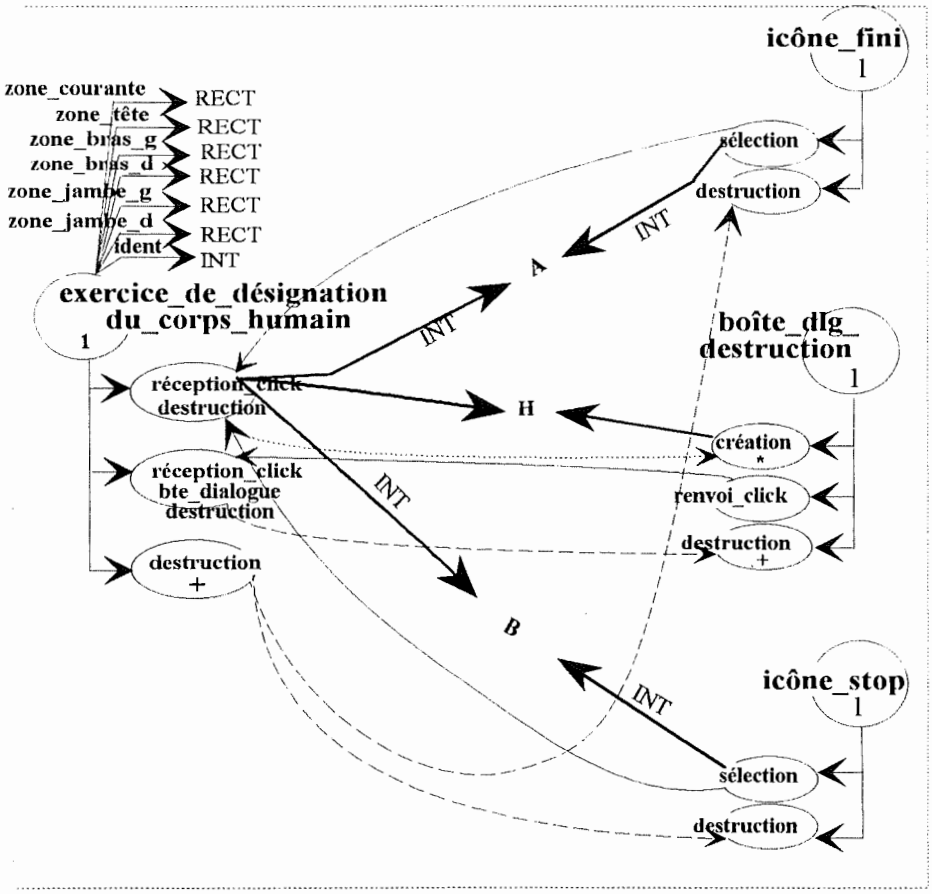
C : est l'ensemble des attributs qui permettront la création de l'objet **Bte_de_dialogue_renforcement.** Cette boîte aura un seul bouton "OK". Les valeurs passées par ces attributs seront :

libellé = "bravo! tu as correctement sélectionné le membre."
localisation = ((70,100),(570,380))
ident = 1
libellé_boutons = "OK"

D: est l'ensemble des attributs nécessaires à la création d'un objet **icône_montre_partie_corps**. Les valeurs passées sont les mêmes que celles indiquées pour l' **icône_montre_partie_corps** créée lors de la création de l'exercice sauf pour l'attribut image qui sera initialisé comme suit :

Si zone_courante = zone_bras_g alors image = "picto_montre_bras_g"
Si zone_courante = zone_bras_d alors image = "picto_montre_bras_d"
Si zone_courante = zone_tête alors image = "picto_montre_tête"
Si zone_courante = zone_jambe_g alors image = "picto_montre_jambe_g"
Si zone_courante = zone_jambe_d alors image = "picto_montre_jambe_d"

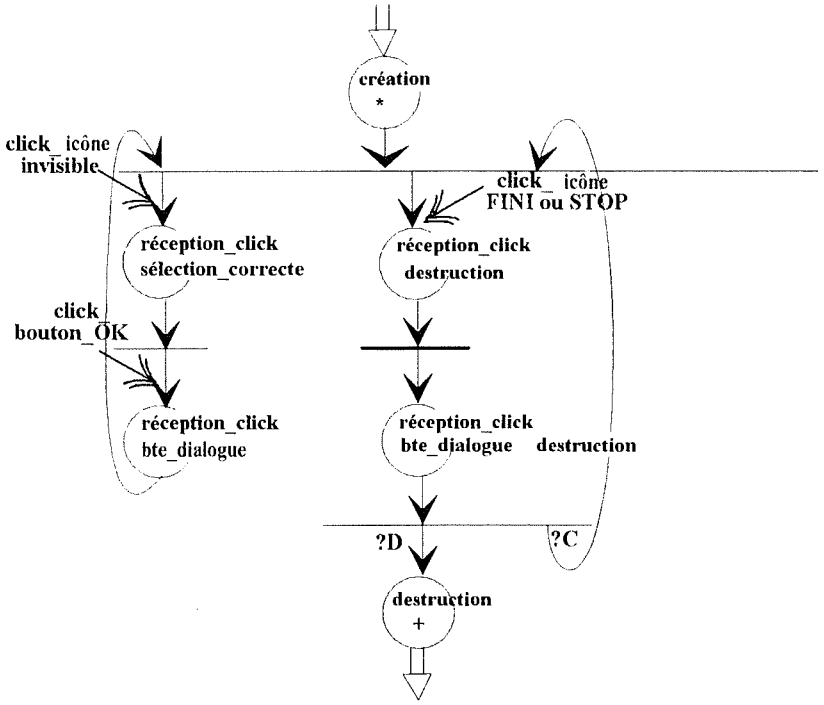
Le schéma suivant modélise le comportement de l'exercice lorsque l'utilisateur désire terminer. Nous n'avons pas modélisé la destruction de tous les objets de l'exercice bien que lorsque l'événement **destruction** de l'exercice soit déclenché, cela se produise. Ceci uniquement pour garder un schéma clair et assez réduit.



H : est l'ensemble des attributs permettant de créer une **boîte_dlg_destruction.** Cette boîte aura deux boutons. Nous avons indiqué précédemment que les traitements associés à l'icône "Fini" et à l'icône "Stop" seraient identiques dans notre modélisation. Cependant, suivant la valeur de l'identifiant passée à **reception_click_destruction**, les libellés des boîtes de dialogue seront différents. Si ident = 100, le libellé sera : "Veux-tu terminer pour aujourd'hui?" tandis que si ident = 101, le libellé sera : "Veux-tu arrêter l'exercice pour l'instant?". L'attribut **libellé_boutons** aura la valeur "OUI-NON" tandis que

les autres attributs auront les mêmes valeurs que celles indiquées lors de la création de l'exercice pour la boîte de dialogue de renforcement.

Diagramme de comportement :



?D ≡ (si id_bouton = 1). Cette condition signifie que l'utilisateur a cliqué sur un bouton de confirmation ou de validation. Le choix de finir l'exercice est donc confirmé.

?C ≡ (Si id_bouton = 2). Le choix de finir l'exercice est infirmé.

Diagramme de mise à jour :

Lors de chaque déclenchement de l'événement **réception click_sélection_correcte,** une mise à jour est effectuée pour assigner à **zone_courante**, les coordonnées de la zone correspondant à la localisation du membre suivant à désigner. Comme les schémas OBLOG prendraient trop de place, nous ferons ces assignations de manière textuelle.

Si zone_courante = zone_tête alors zone_courante = zone_bras_g
Si zone_courante = zone_jambe_d alors zone_courante = zone_tête
Si zone_courante = zone_jambe_g alors zone_courante = zone_jambe_d
Si zone_courante = zone_bras_g alors zone_courante = zone_bras_d
Si zone_courante = zone_bras_d alors zone_courante = zone_jambe_g

5.2. Exercice de placement du corps humain

Cet exercice est tiré du cahier des charges qui nous a été fourni par les deux éducatrices du "Trèfle" de Kain, Madame Van Diependael et Madame Quintin.

Nous l'avons repris pour pouvoir montrer une partie des traitements que notre objet icône peut réaliser. Dans le cadre de cet exercice, c'est surtout sa propriété de mobilité qui est mise en évidence.

5.2.1. Scénario de l'exercice

Cet exercice sert à apprendre à la personne handicapée à positionner correctement les divers parties d'un corps humain les unes par rapport aux autres. Ce n'est plus un exercice de "désignation" mais bien de "puzzle". Nous aurons un tronc humain fixe sur lequel cinq parties amovibles seront à placer correctement par l'utilisateur (la tête et les membres). Chaque fois qu'il aura placé correctement une partie du corps, une boîte de dialogue de renforcement apparaîtra. Il lui suffira alors de valider la prise de connaissance du message et il pourra continuer l'exercice. Pour indiquer qu'il a terminé, il faudra qu'il clique sur l'icône "FINI". Nous pouvons observer l'interface de cet exercice à la figure 5.2.

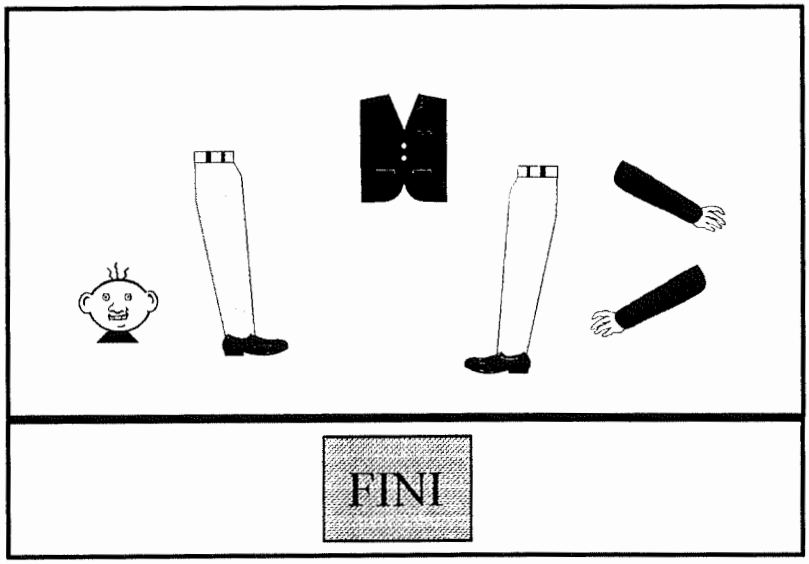


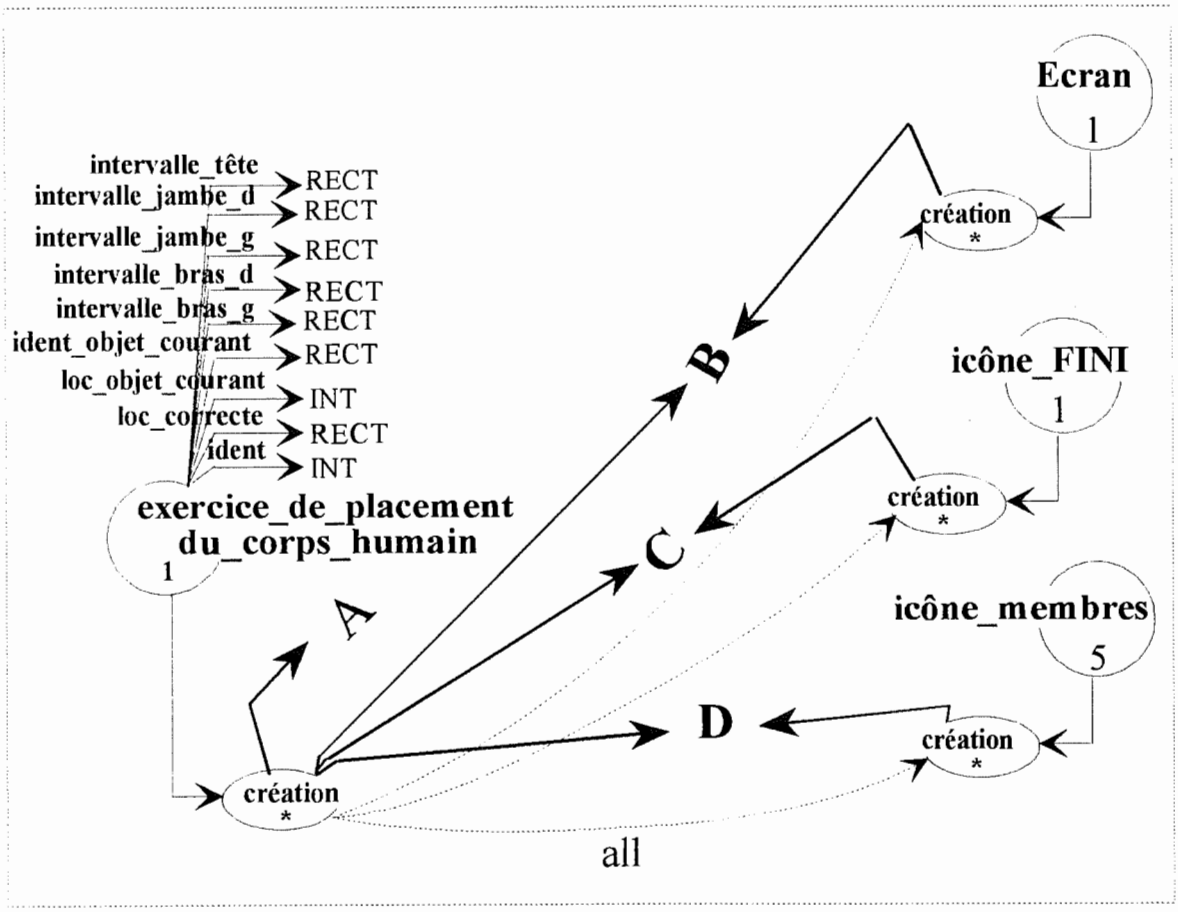
Figure 5.2. : interface de l'exercice de placement du corps humain

5.2.2. Modélisation de l'exercice

Pour la modélisation, nous avons besoin d'un objet de type écran avec une zone de commande dans laquelle nous positionnerons une icône sélectionnable et visible de fin

d'exercice. Dans la zone de travail de cet écran se trouvera un pictogramme représentant un tronc humain. Nous retrouverons également, à des coordonnées aléatoires, cinq icônes sélectionnables, amovibles et visibles illustrées par des pictogrammes représentant les parties du corps.

Diagramme matriciel et d'interaction :



A : constitue l'ensemble des données nécessaires pour créer l'exercice. Il s'agit des attributs de l'objet **Exercice_de_placement_du_corps_humain**, mais également des attributs nécessaires à la création des divers objets composants.

Les valeurs suivantes seront passées et initialiseront les attributs de l'objet :

intervalle_tête = ((244,12),(284,32))
intervalle_bras_g = ((343,115),(383,135))
intervalle_bras_d = ((148,115),(188,135))
intervalle_jambe_g = ((319,245),(359,265))
intervalle_jambe_d = ((229,245),(269,265))
ident = 1

B : constitue l'ensemble des attributs nécessaires à la création de l'objet **Ecran.** Les valeurs passées seront :

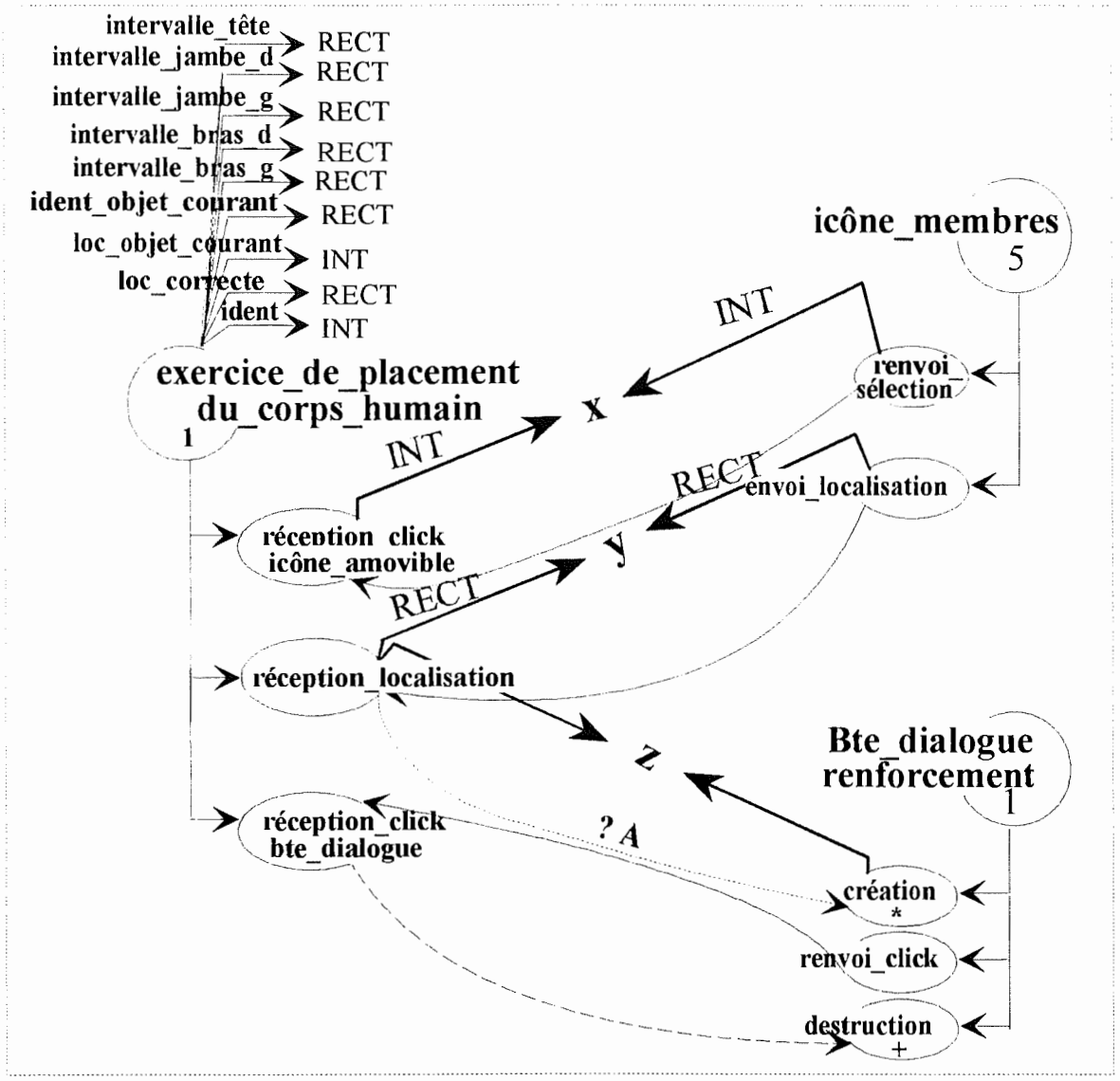
ident = 1
hauteur_zone_cde = 90
couleur_fd_cde = blanc
couleur_fd_travail = blanc
image_fond = "tronc_humain"

C : constitue l'ensemble des attributs nécessaires à la création de l'objet **icône_fini.** Les valeurs passées seront :

ident = 100
image = "picto_fini"
couleur_fond = vert
couleur_bord = noir
localisation = ((288,10),(352,74))
visible = vrai
amovible = faux
pile = faux
sélectionnable = vrai

D : constitue l'ensemble des attributs nécessaires à la création des objets **icône_membres.** Les valeurs passées seront respectivement :

ident = 1^2^3^ 4^ 5
image = "picto_bras_g" ^ "picto_bras_d" ^ "picto_tête" ^ "picto_jambe_g" ^ "picto_jambe_d"
couleur_fond = blanc
couleur_bord = blanc
localisation = ((363,125),(558,155)) ^ ((168,125),(323,155)) ^ ((264,22),(284,52)) ^ ((339,255),(364, 355)) ^ ((249,255),(254,355))
visible = faux
amovible = vrai
pile = faux
sélectionnable = vrai



?A : si (loc_objet_courant ⊂ localisation_correcte)

Z : est l'ensemble des attributs qui permettront la création de l'objet **Boîte_de_dialogue_renforcement**. Cette boîte aura un seul bouton. Les valeurs passées pour ces attributs seront :

libellé = "bravo! tu as correctement placé la partie du corps."
localisation = ((70,100),(570,380))
ident = 1
libellé_boutons = "OK"

Diagramme de mise à jour :

L'événement **réception_click_icône_amovible** met à jour :

ident_objet_courant = X

L'événement **réception_localisation** met à jour :

loc_objet_courant = Y

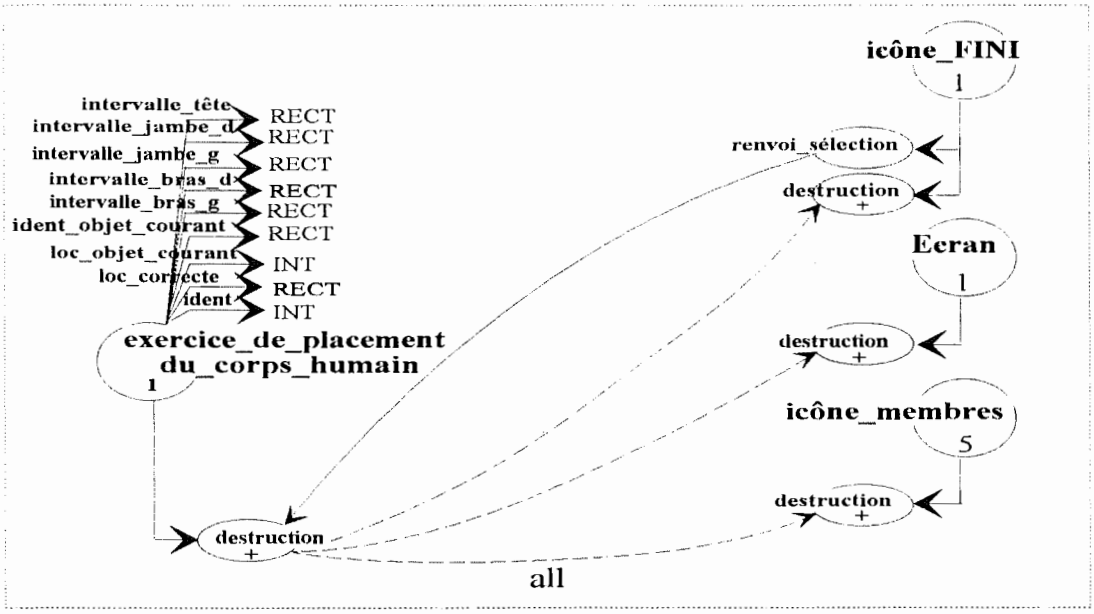
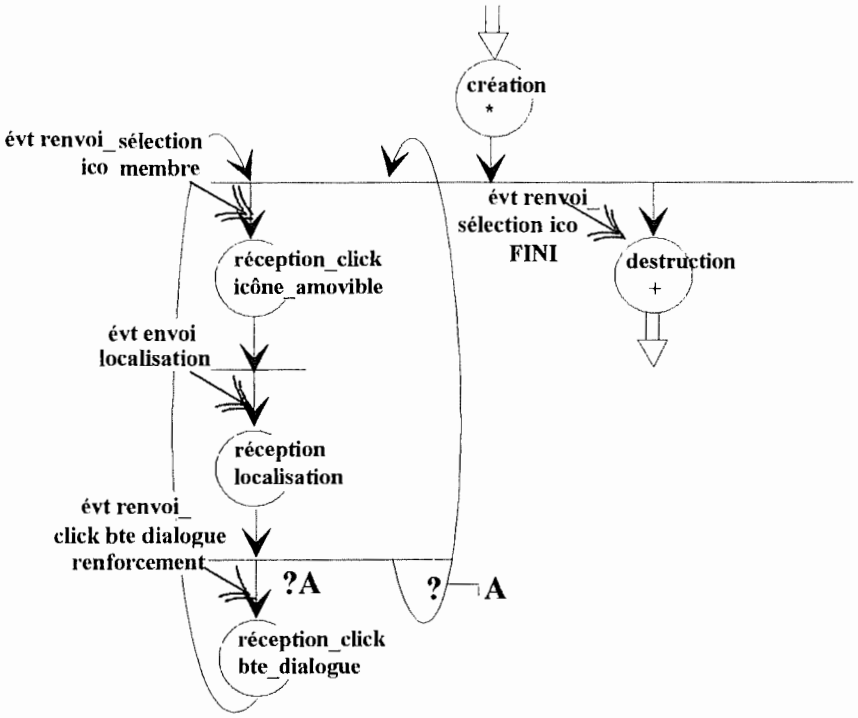


Diagramme de comportement :



? A : si localisation_objet_courant ⊂ localisation_correcte.

5.3 Interface de sélection des desserts

Nous allons à présent modéliser une partie de l'interface du logiciel d'aide à la gestion des achats alimentaires. "Ce logiciel permet à l'utilisateur d'établir une liste d'achats à effectuer sur base de la liste des ingrédients entrant dans la composition de plats composant des menus constitués par l'utilisateur et sur base d'une liste de produits à toujours avoir sous la main" [Tope, 91].

5.3.1 Scénario de la sélection de desserts

Nous avons choisi de nous attacher à la partie du logiciel qui permet à l'utilisateur de constituer des menus. Nous avons modélisé la partie de l'interface permettant de choisir un dessert. Ceci parce que l'utilisateur a le choix entre plusieurs "éléments de menus". C'est donc un exercice permettant de montrer l'opportunité de l'utilisation d'un dérouleur. L'interface en est montrée a la figure 5.3.

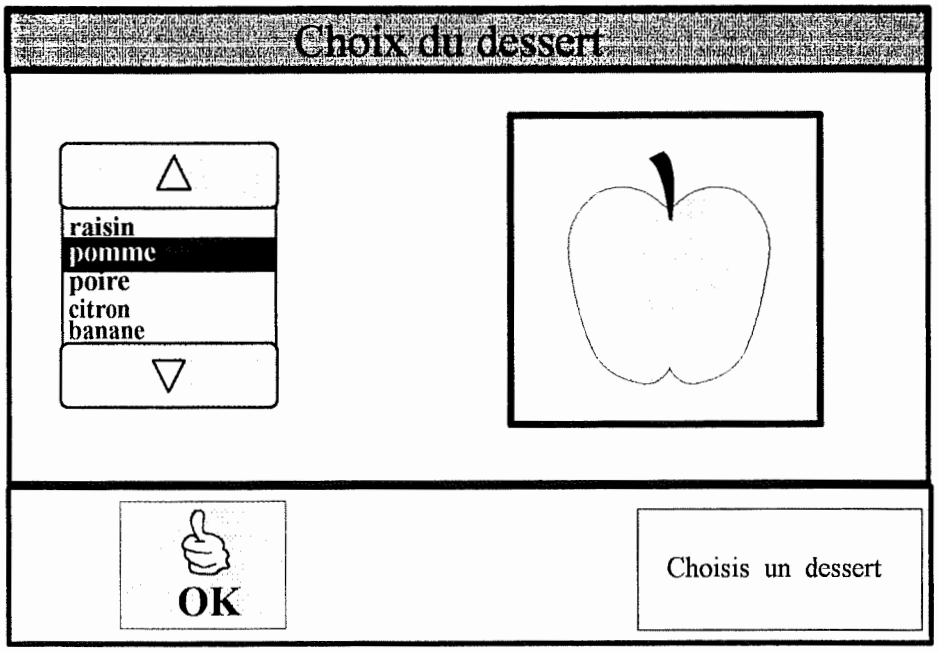


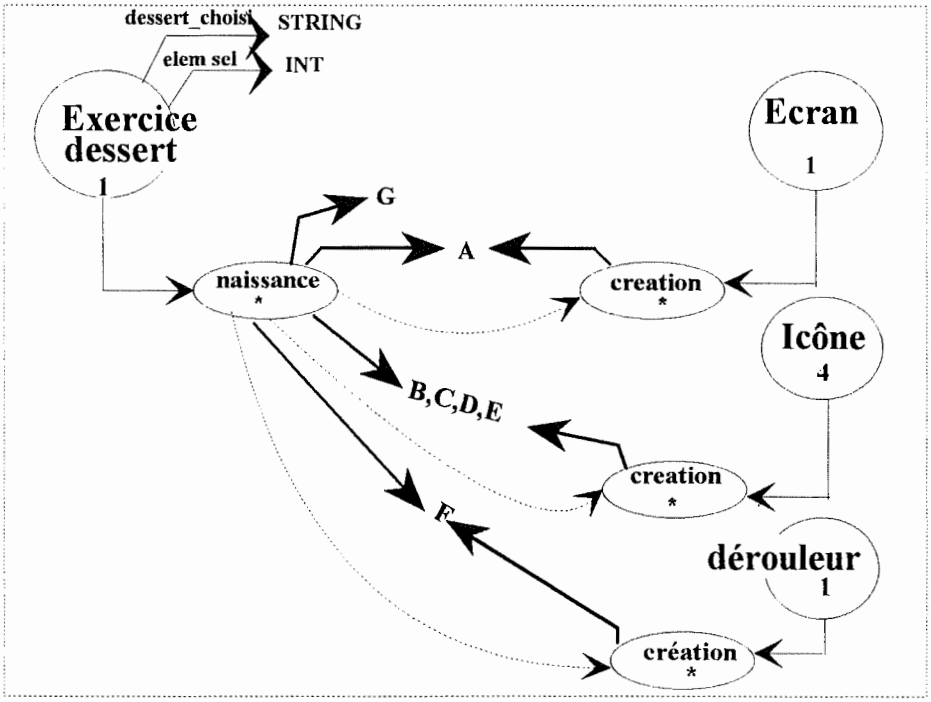
Figure 5.3. : interface de l'écran de désignation d'un dessert

5.3.2 Modélisation de l'interface

Cette partie de l'interface est réalisable au moyen des objets interactifs offerts décrits dans le chapitre précédent. Elle sera constituée d'un dérouleur textuel à sélection par bouton qui permettra à l'utilisateur de visualiser les pictogrammes des différents desserts

avant de fixer son choix. Elle nécessitera, de plus, l'utilisation de trois icônes non sélectionnables pour le titre de l'écran, la représentation pictographique de l'élément du dérouleur couramment sélectionné et pour la phrase explicative de l'écran : "Choisis un dessert". Un objet icône sélectionnable de validation sera utilisé. Il permettait, dans le travail de L. Topet et X. Vonèche, de fixer le choix du dessert et de passer à la suite du menu. Nous nous limiterons ici à la modélisation du choix des desserts. En conséquence, la sélection de l'icône de validation aura pour effet de fixer le choix du dessert et de terminer l'exercice.

Diagramme matriciel et d'interaction :



G : reprend l'ensemble des attributs qui sont passés à l'événement de création.

A : ensemble des valeurs d'attributs nécessaires à la création d'**Ecran.**
Couleur_fd_travail = azur,
couleur_fd_cde = rouge,
hauteur_zone_cde = 96 (pixels),
ident = 1,
image_fond = NULL,

B : ensemble des valeurs nécessaires à la création de l'icône **Titre_écran**.
Amovible = faux,
visible = vrai,
pile = faux,

selectionnable = faux,
localisation = ((0,0),(64,70)) (pixels),
couleur_bord = blanc,
couleur_fond = vert,
ident = 1,
image = "titre écran" (ce sera en fait la référence vers le pictogramme),

C : Ensemble des valeurs des attributs nécessaires à la création de l'icône illustré par la phrase "**choisis un dessert**". Pour les mêmes attributs, les valeurs sont identiques aux précédentes sauf pour :

localisation = ((320,300),(640,390)),
couleur_bord = noir,
couleur_fond = gris clair,
ident = 2,
image = "Choisis dessert".

D : Ensemble des valeurs des attributs nécessaires à la création de l'icône **Main**. Les valeurs sont identiques sauf pour :

sélectionnable = vrai,
localisation = ((288,464),(352,464)),
couleur_bord = noir,
couleur_fond = jaune,
ident = 3,
image = "Main".

E : Ensemble des valeurs des attributs nécessaires à la création de l'icône représentatif du **Dessert.** Les valeurs des attributs sont identiques sauf pour :

sélectionnable = faux,
localisation = ((328,75),(480,280)),
couleur_bord = noir,
couleur_fond = azur,
ident = 4,
image = "icône vide". Cette icône n'aura pas de pictogramme associé lors de l'initialisation car nous avons vu que l'élément constrasté d'un dérouleur était vide lorsque celui-ci apparaît à l'écran. L'icône sera donc constituée uniquement de la couleur de fond.

F : Ensemble des valeurs nécessaires pour l'initialisation du dérouleur.
Nbre_elem = 5,
localisation = ((10,70),(300,300)),

ident = 1,

les attributs communs des éléments de dérouleur :

type = libellé,

couleur_fd_contrast = noir,

couleur_text_contrast = blanc,

couleur_fd = vert,

les attributs particuliers des éléments de dérouleur :

lib[1] = "raisin", lib[2] = "pomme", lib[3] = "poire", lib[4] = "citron", lib[5] = "banane",

les identifiants vont de 1 à 5,

les attributs relatifs aux flèches de défilement :

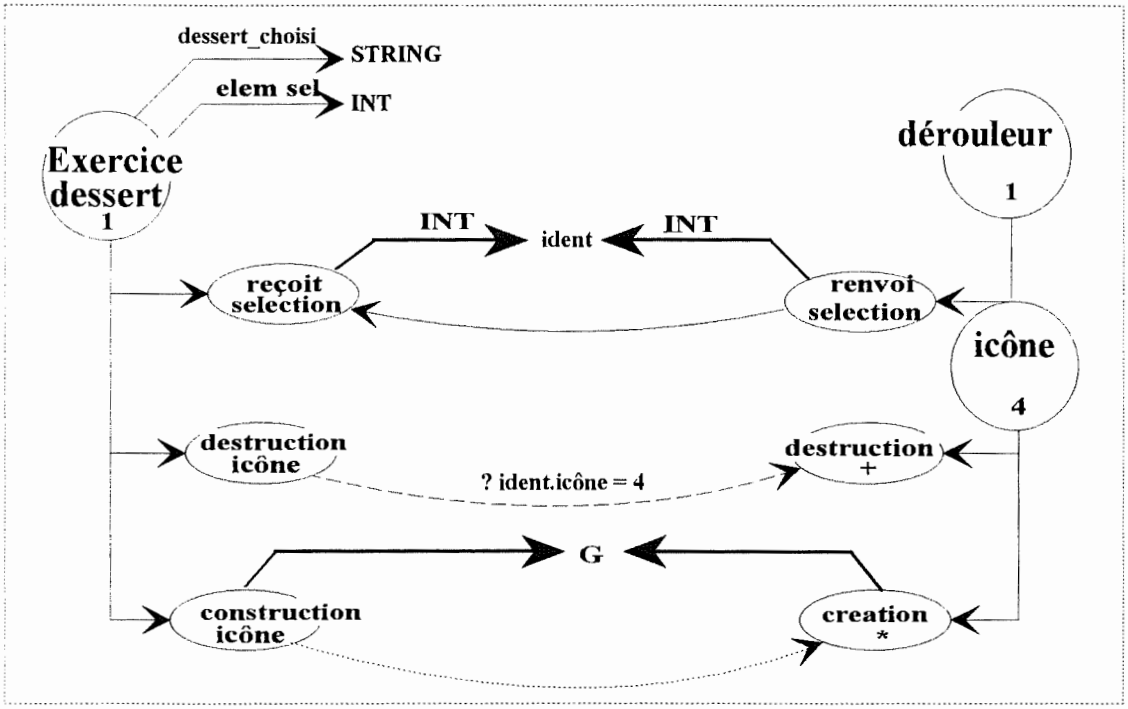
hauteur = 25,

largeur = 290,

couleur_fd = jaune,

couleur_lib = noir,

ident1 = -1 et ident2 = 1, ces deux identifiants seront assignés automatiquement aux flèches de défilement.



G : Est l'ensemble des valeurs des attributs passées à l'événement de création de l'icône représentatif du **Dessert**. Ces dernières sont identiques à celles passées lors de la création de l'exercice sauf pour :

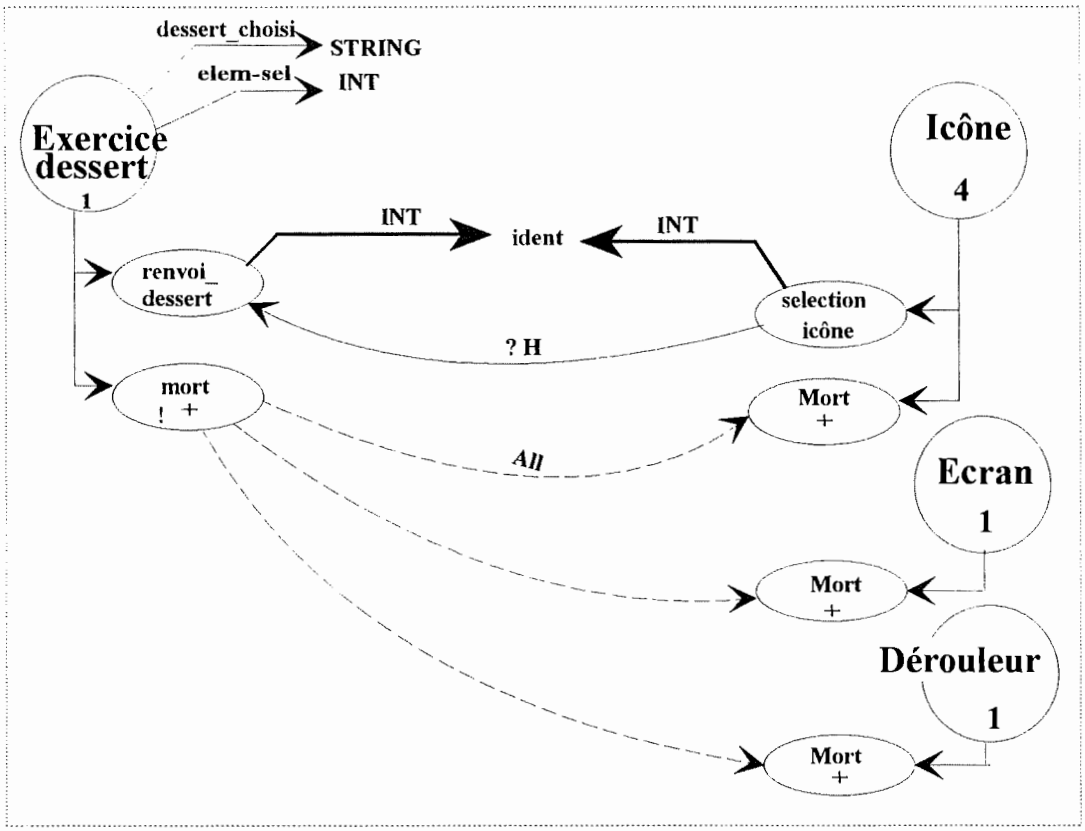
image = dessert_choisi.

dessert_choisi est mis à jour par l'événement **construction icône** comme indiqué ci-dessous.

si elem_sel = 1 alors dessert_choisi = "raisin",
si elem_sel = 2 alors dessert_choisi = "pomme",
si elem_sel = 3 alors dessert_choisi = "poire",
si elem_sel = 4 alors dessert_choisi = "citron",
si elem_sel = 5 alors dessert_choisi = "banane".

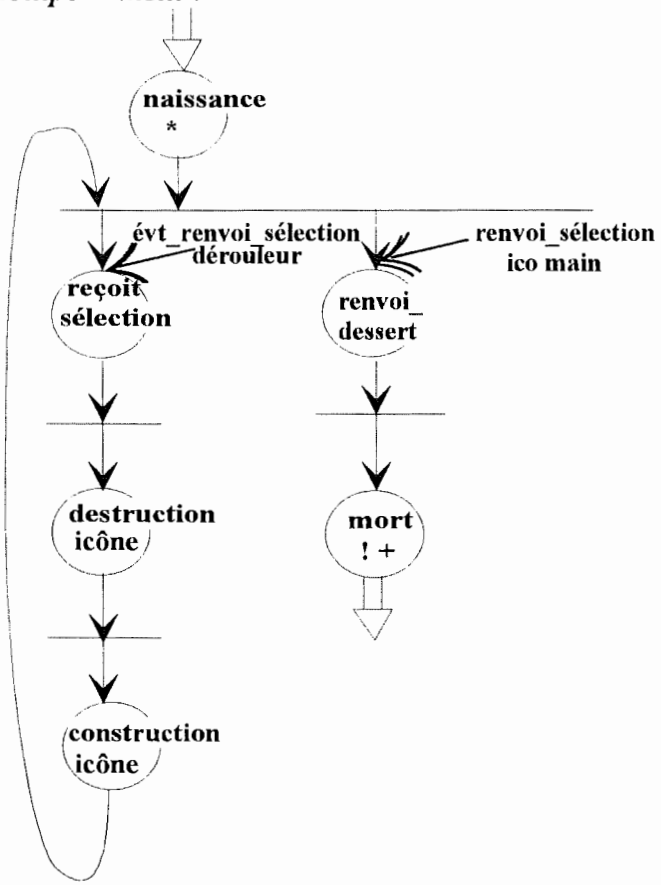
En outre, **elem_sel** est lui-même mis à jour par l'événement **reçoit sélection** qui reçoit la valeur par le dérouleur.

elem_sel = X



? H : ident = 3. Ici, il ne pourrait avoir une autre valeur car l'icône d'identifiant 3 est le seul qui soit sélectionnable.

Diagramme de comportement :



5.4. Exercice d'élaboration de suites d'objets

Présentons ci-dessous un exercice du cahier des charges reçu des éducatrices du Trèfle. C'est un exercice de réalisation de suite. Le personne handicapée devra sélectionner et positionner certain objets graphiques pour former une suite d'objets, en alternance des ronds et des carrés. L'exercice se présentera comme à la figure 5.4.

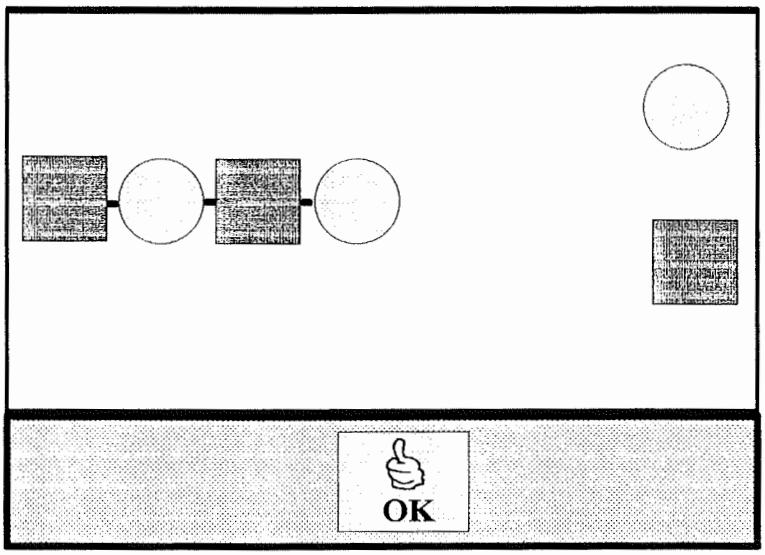


Figure 5.4. : interface de l'exercice d'élaboration d'une suite

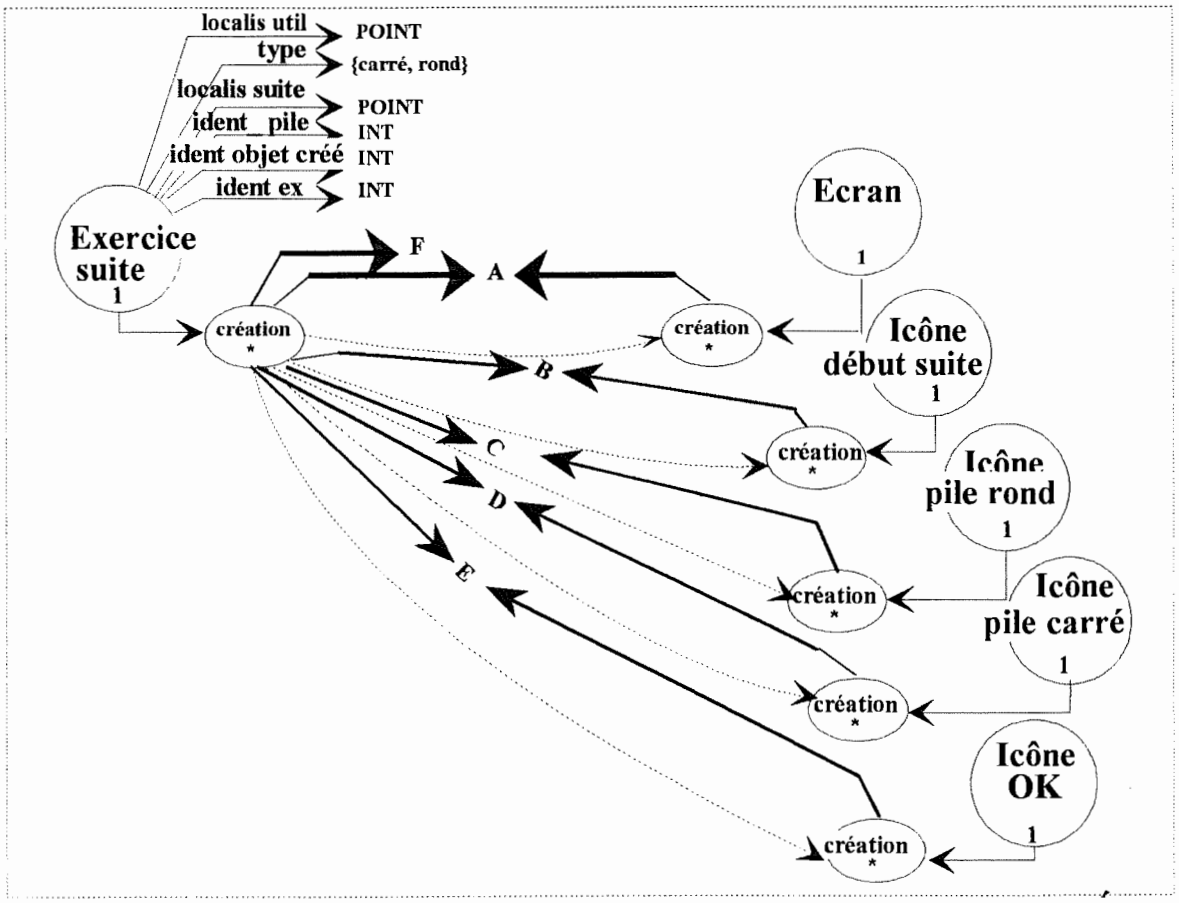
5.4.1. Scénario de l'exercice

Lorsque l'utilisateur déposera son objet près de la suite, une zone d'imprécision de 64 pixels sur 126 est acceptée. Ceci afin de ne pas décourager l'utilisateur ayant des difficultés lors de la manipulation de la souris. Si l'objet est placé dans cette zone, il sera ajusté correctement à la fin de la suite. En outre, un objet ne complétant pas correctement la suite, même s'il est placé dans cette zone, disparaîtra de l'écran afin de ne pas l'encombrer inutilement.

5.4.2. Modélisation de l'exercice

La partie de la suite présentée à l'écran en guise d'exemple pour l'utilisateur sera une icône insélectionnable. Les deux éléments disponibles pour compléter la suite seront, quant à eux, des icônes ayant la propriété pile. En effet, une représentation des éléments constitutifs de la suite doit toujours être disponible dans la partie droite de la zone de travail de l'écran. En outre, une icône de validation permettra de terminer l'exercice. Cet objet doit bien entendu être sélectionnable.

Diagramme matriciel et d'interaction :



F reprend l'ensemble des valeurs des attributs qui sont passées à l'événement de **création.** Il s'agit des valeurs de certains attributs de l'**exercice suite** et des valeurs permettant de créer les objets de l'exercice.

L'événement de **création** initialisera **localis util** à (232,191) et **type** à "carré".

A : ensemble des valeurs permettant de créer l'**écran**.
couleur_fd_travail = azur,
couleur_fd_cde = rouge,
hauteur_zone_cde = 96,
ident = 1,
image_fond = NULL.

B : ensemble des valeurs permettant de créer l'**icône début suite.**
amovible = faux,
visible = vrai,
pile = faux,
selectionnable = faux,
localisation = ((20,170),(232,212)),
couleur_bord = blanc,
couleur_fond = azur,
ident = 1,
image = "début suite"

C : ensemble des valeurs permettant de créer l'**icône pile carré**. Pour les mêmes attributs, les valeurs sont identiques aux précédentes sauf pour :
pile = vrai,
sélectionnable = vrai,
localisation = ((578,42),(620,82)),
ident = 2,
image = "carré".

D : ensemble des valeurs permettant de créer l'**icône pile rond**. Pour les mêmes attributs, les valeurs sont identiques à celles de **C** sauf pour :
localisation = ((578,382),(620,424)),
ident = 3,
image = "rond".

E : ensemble des valeurs permettant de créer l'**icône OK.** Elles sont identiques à celles de **B** sauf pour :

sélectionnable = vrai,

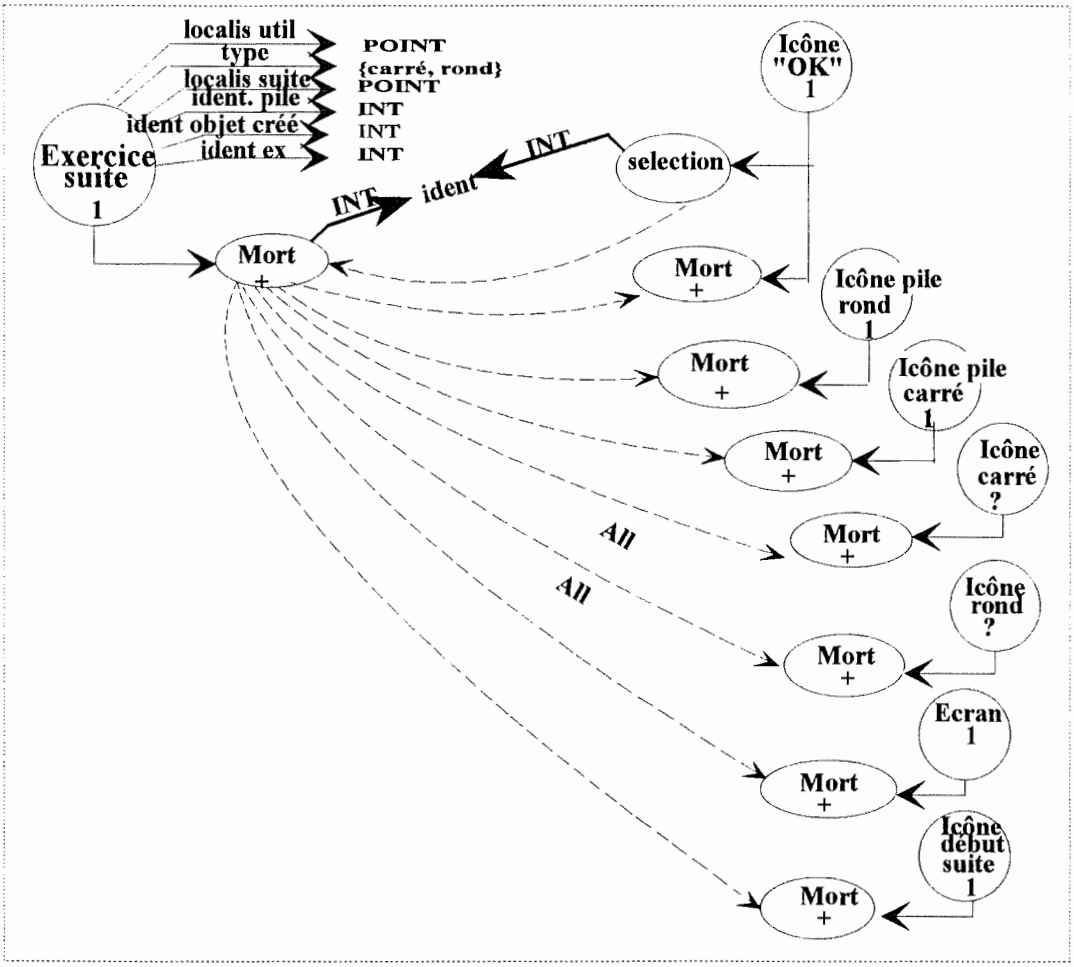
localisation = ((290,402),(350,462)),

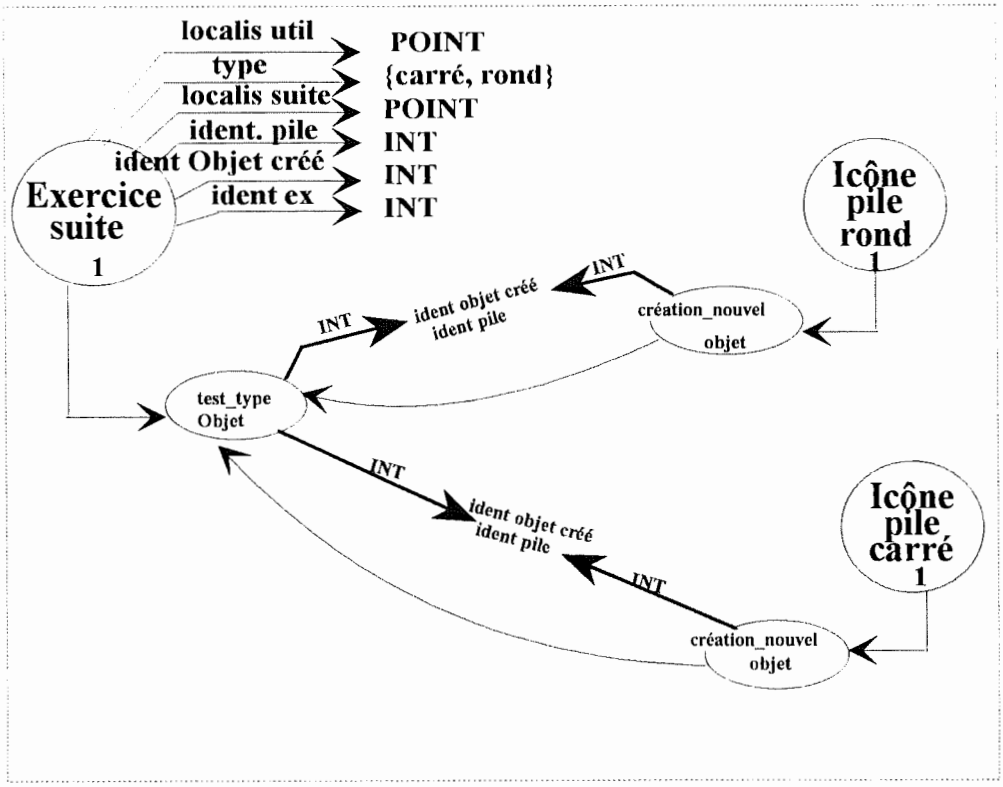
couleur_bord = noir,

couleur_fond = vert,

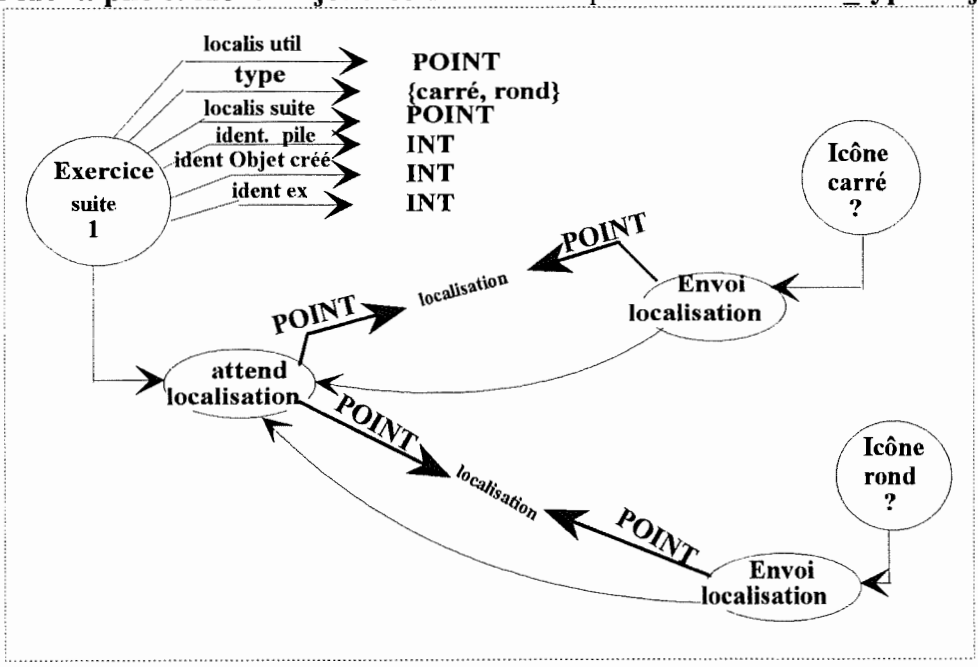
ident = 4,

image = "OK".

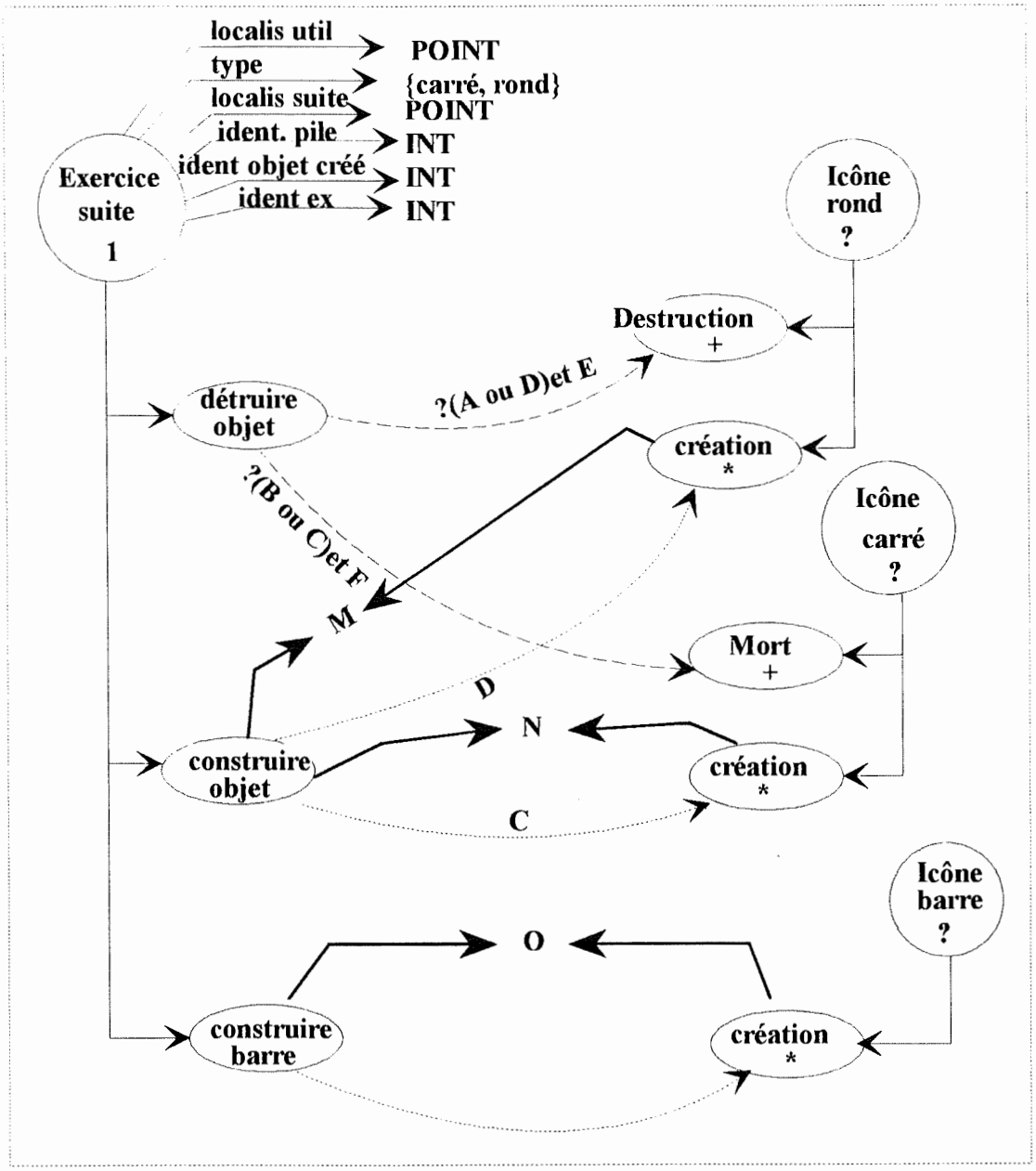




Les valeurs passées par l'événement **création_nouvel objet** sont stockées dans les attributs **ident. pile** et **ident Objet créé** de l'exercice par l'événement **test_type Objet.**



L'événement **attend localisation** met à jour l'attribut **localis util** de l'exercice grâce aux valeurs reçues par l'événement **envoi localisation** de l'icône créée.



? A : type = "carré" et ident. pile = 3.

? B : type = "rond" et ident. pile = 2.

? C : type = "carré" et ident.pile = 2.

? D : type = "rond" et ident.pile = 3.

? E : indent objet créé.exercice suite = ident.icône rond.

? F : indent objet créé.exercice suite = ident.icône carré.

L'événement **détruire objet** demande la destruction de l'objet dont l'identifiant est dans l'attribut **ident Objet créé** de l'**exercice suite**.

L'événement **construire objet** initialise les valeurs des attributs d'une icône à créer comme suit :

amovible = faux,

visible = vrai,

pile = faux,

sélectionnable = faux,

localisation = ((**localis suite**.x, **localis suite**.y - 21),(**localis suite**.x + 42, **localis suite**.y + 21)),

couleur_bord = blanc,

couleur_fond = azur,

ident = **ident Objet créé**,

image = "rond" dans M, "carré" dans N.

De plus, il met à jour les attributs de l'**exercice suite** comme suit :

localis suite = (localis suite.x + 42, localis suite.y),

type = " rond" si la valeur précédente était carré, "carré" sinon.

L'événement **construire barre** initialise les attributs d'une icône comme **construire objet** sauf pour :

localisation = ((localis suite.x, localis suite.y - 21), (localis suite.x + 11, localis suite.y + 21)),

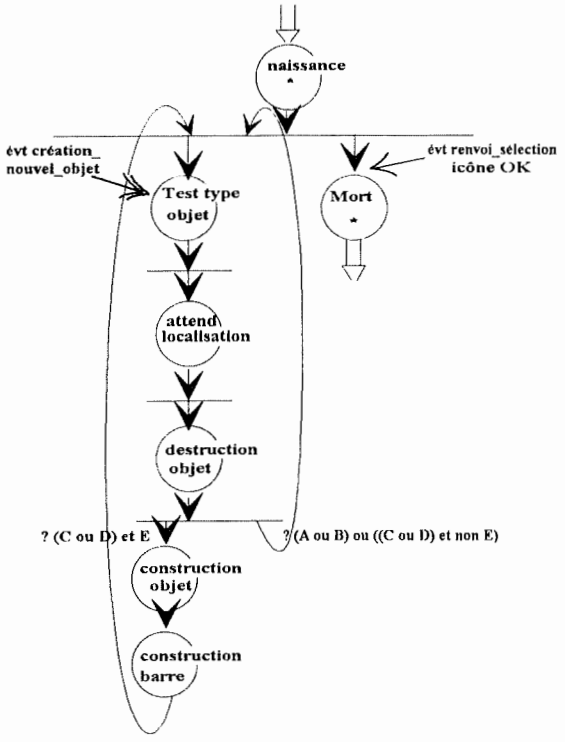
ident = **ident Objet créé** + 1,

image = "barre",

Il met à jour l'attribut **localis.suite** comme suit :

localis suite = (localis suite.x + 11, localis suite.y)

Diagramme de comportement :



? A : type = "carré" et ident. pile = 3.

? B : type = "rond" et ident. pile = 2.

? C : type = "carré" et ident.pile = 2.

? D : type = "rond" et ident.pile = 3.

? E : localis suite.x - 32 < localis util.x < localis suite.x + 32. et localis suite.x - 63 < localis util.y < localis suite.x + 63.

Conclusion

Comme nous pouvons le constater, la portée des traitements qui peuvent être effectués à l'aide de nos objets est considérable, bien qu'incomplète. Cet ensemble d'objets peut être étendu à volonté.

Chapitre 6 : Windows : sa structure, son fonctionnement et son influence sur notre travail

Introduction

Windows, la plus célèbre interface graphique du monde du P.C., est l'enfant de la firme Microsoft Inc. Née en 1985, elle a subi de nombreuses évolutions pour devenir à l'heure actuelle l'interface graphique de référence pour plus de 10 millions d'utilisateurs.

Toute l'implémentation des objets graphiques spécifiés précédemment a été réalisée sous cet environnement. Comme nous l'avons déjà mentionné, ce n'est pas une mince tâche que de se lancer dans la programmation d'interfaces graphiques sans une expérience préalable de ce domaine. Notre inexpérience de ce genre de programmation ainsi que notre méconnaissance du langage C ont constitué de sérieux freins au cheminement qui nous a conduits à la réalisation de ce mémoire.

Nous tenons, par ce présent chapitre, à vous présenter MS-Windows, ses caractéristiques, son fonctionnement mais également les embûches de sa programmation et ses imperfections. Nous consacrerons également une section à montrer que, même si l'approche orientée objet n'est pas aussi bien supportée que sur certains autres environnements, elle a considérablement influencé la structure de Windows ainsi que celle de ses applications.

6.1. Windows

MS-Windows est un système d'exploitation qui fonctionne sur toute la gamme des P.C.. Ce système tourne et a été conçu sous MS-DOS.

Néanmoins, Windows constitue une couche, qui peut être complètement hermétique, au-dessus de MS-DOS. Cela signifie qu'il offre une palette de fonctionnalités tellement riche et variée, que toutes les applications développées sous cet environnement peuvent ignorer les fonctions offertes par MS-DOS.

Windows offre des fonctionnalités de gestion de mémoire, de gestion de fichiers, de gestion d'affichage sur écran ou imprimante et de gestion des entrées par divers moyens d'interaction. L'utilisateur peut donc disposer d'un ensemble de fonctions qui couvre tous les aspects de l'exploitation de la machine. C'est pourquoi il est courant d'assigner à Windows le nom de **machine virtuelle**. En effet, lorsque nous travaillons dans l'environnement Windows, nous avons l'impression d'être sur une autre machine.

6.1.1. Son fonctionnement

Le fonctionnement de Windows est souvent considéré, par la plupart des utilisateurs et même par certains programmeurs, comme énigmatique.

En effet, la littérature courante donne une vision conceptuelle du fonctionnement de Windows qui semble tirer sa pertinence plus de sa facilité de compréhension que de sa rigueur fonctionnelle. Nous donnerons donc, dans cette section, une vision du fonctionnement de Windows qui semble réunir autour d'elle un certain consensus.

Windows offre divers services aux programmeurs qui utilisent ses fonctionnalités. Il permet l'exécution de fonctions d'Entrée/Sortie (E/S) vers les périphériques, la gestion de la mémoire et le support du traitement multitâche.

Tous ces services offerts sont répartis dans divers composants de l'environnement Windows constituant un ensemble de "**gestionnaires**". Nous retrouvons souvent dans la littérature des références au "gestionnaire de tâches", au "gestionnaire de la mémoire" ainsi qu'au "gestionnaire de fenêtres". Ces "gestionnaires" ne sont pas des programmes manipulables comme n'importe quelle application Windows. Ils n'existent que pour pouvoir donner une vue fonctionnelle de Windows. Cette vue possède encore l'avantage de correspondre, mais de manière assez éloignée, aux divers modules qui constituent l'environnement Windows.

Comme nous l'avons vu à la section 2.3.2., Les fonctionnalités offertes par Windows sont réparties en divers modules. Ces modules constituent l'**Application Programming Interface** (API) de Windows.

Les trois principaux modules sont le **Graphics Device Interface** ou **GDI**, le **User** et le **Kernel, Krnl286** ou **Krnl386** qui sont des Bibliothèques à Liens Dynamiques (BLD). Les fonctions restantes, principalement utilisées pour l'accès aux périphériques d'E/S, sont implémentées dans un ensemble restreint de "gestionnaires de périphériques". Tous ces modules ainsi que leur rôle dans Windows sont décrits dans le tableau 6.1.

Nom du module	***Fonctions supportées***
GDI.EXE	Interface graphique : sortie des images graphiques, gestion de la palette de couleurs
USER.EXE	Fenêtre, icône et gestion du curseur
KERNEL.EXE	Gestion de la mémoire, de la liste de tâches
KRNL286.EXE	" " "
KRNL386.EXE	" " "

COMM.DRV	Gestion de périphériques de communication asynchrone
KEYBOARD.DRV	Gestion du clavier
MOUSE.DRV	Gestion de la souris
SOUND.DRV	Gestion du générateur de sons
SYSTEM.DRV	Gestion du "timer", unités de disque

Tableau 6.1. : Modules constitutifs de Windows

Cet ensemble de gestionnaires permet à Windows d'acquérir une très forte indépendance hardware. Le système peut en effet fonctionner sous un nombre assez impressionnant de configurations diverses. Il supporte pratiquement n'importe quelle imprimante, des écrans de résolutions diverses, un certains nombre de cartes son, ...

Le programmeur d'application Windows n'a cependant pas besoin de prêter attention à ces différents modules. Il peut appeler, quand bon lui semble, les diverses fonctions qui y sont supportées, sans se soucier des modules qui les contiennent.

A. Le gestionnaire de tâches

Comme nous l'avons déjà mentionné précédemment, Windows supporte le traitement multitâche.

Dans le monde du P.C., nous retrouvons deux types de traitements multitâches. Le schéma ***multitâche préemptif***, où le système d'exploitation accorde un temps prédéterminé à chaque tâche et passe le contrôle de tâche à tâche indépendamment du fait que les tâches rendent ou ne rendent pas la main au système. C'est dans cette catégorie que nous retrouvons notamment le nouveau système d'exploitation d'IBM, l'OS/2 2.0.

Windows, quant à lui, supporte un traitement ***multitâche coopératif***. Ici, le système transfère le contrôle à une tâche et tant que celle-ci ne rend pas la main au gestionnaire de tâches, aucune autre tâche ne peut s'exécuter. Notons que Windows considère chaque instance d'une application comme une tâche différente.

Malheureusement, toute médaille ayant son revers, le traitement multitâche impose également certaines contraintes.

Dans toute application Windows, les données sont séparées des traitements. Ceci pour la simple et bonne raison que Windows charge en mémoire, pour chaque instance d'un module d'un programme, une copie du segment de données par défaut, alors que le code

exécutable lui, n'est chargé qu'une seule fois. Cette technique permet notamment des gains de mémoire considérables.

Une autre contrainte qui est imposée par le traitement multitâche concerne les ***fonctions fenêtres*** de chaque application Windows; une fonction fenêtre étant la composante de gestion du comportement d'une fenêtre. Lors d'une section suivante, nous examinerons plus en détail la gestion des fenêtres. Pour que le système puisse passer le contrôle à une application, il faut que celle-ci soit conçue avec au moins une fonction fenêtre qui puisse être appelée. Afin que cela puisse se réaliser, il faut que les diverses fonctions fenêtres soient exportées lors de la compilation et du chaînage du programme. De cette façon, Windows ou un autre programme Windows, peut importer, appeler ces diverses fonctions. Ceci est extrêmement important car c'est la base même du fonctionnement de Windows.

Windows est lui aussi constitué d'un certain nombre de fonctions qu'il exporte. Lorsqu'il appelle une application, c'est généralement pour lui signaler qu'un événement s'est produit et nécessite un traitement de la part du programme. Pour effectuer ce traitement, l'application va faire appel aux fonctions exportées par Windows. Le flux des échanges de données entre le système et une application est repris à la figure 6.1.

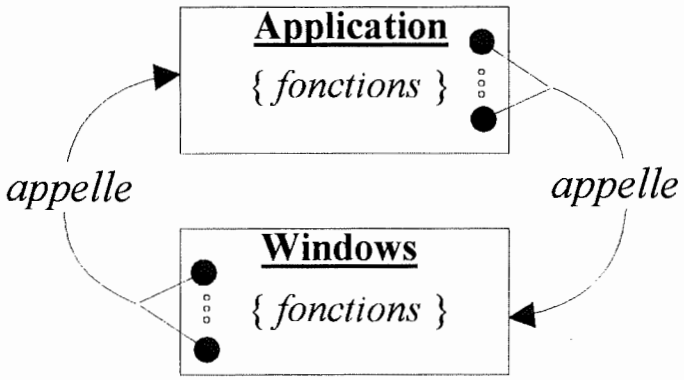


Figure 6.1. : Flux des échanges de données.

Ce schéma montre que l'exécution d'une application n'est qu'une suite d'appels de fonctions entre cette dernière et Windows. Appels qui permettent, tant au système qu'à l'application, de s'expédier mutuellement des messages. Nous reviendrons de manière plus détaillée sur la communication entre une application et le système, dans la section traitant de la structure des applications Windows.

B. Le gestionnaire de fenêtres

La gestion des fenêtres ne consiste pas uniquement à effectuer le traitement graphique nécessaire à l'affichage de la fenêtre à l'écran.

En Windows, chaque fenêtre est composée d'une structure de donnée détenue et mise à jour par le gestionnaire de fenêtres ainsi que d'une fonction fenêtre qui permet de traiter tous les messages dont elle est destinataire.

Une fenêtre est en fait une unité de traitement qui n'aboutit pas nécessairement à un résultat à caractère visuel.

Prenons par exemple, la liste des tâches. Comme nous l'avons vu dans le point précédent, c'est le gestionnaire des tâches qui s'occupe de gérer l'exécution des applications. Tout ce travail est effectué par le biais d'une fenêtre qui n'apparaît pas systématiquement à l'écran. Pour faire apparaître cette fenêtre, il faut réduire tous les programmes courants sous forme iconique et effectuer un double click dans l'espace de travail. La fenêtre contenant la liste des tâches en cours apparaît alors comme à la figure 6.2.

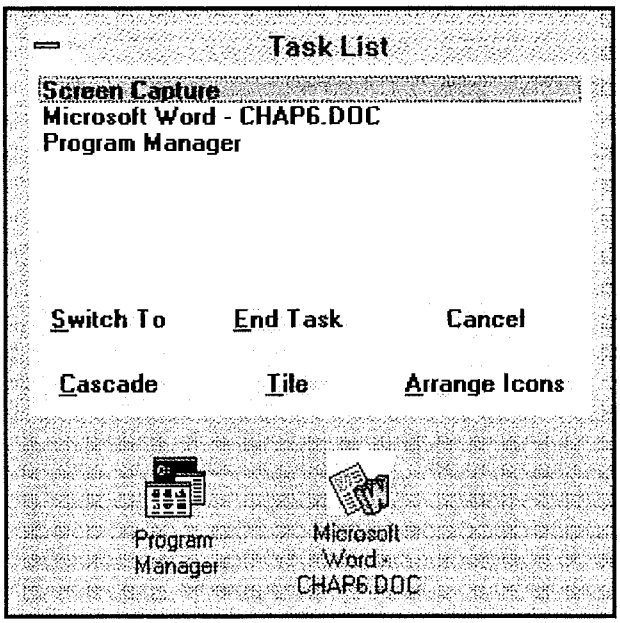


Figure 6.2. : fenêtre de la liste des tâches

Une application Windows peut donc créer une fenêtre qui aura pour seul objectif le traitement d'un certain nombre de messages, sans pour cela apparaître à l'écran sous une forme quelconque.

Chaque programme peut, bien entendu, créer une multitude de fenêtres que le gestionnaire va gérer sous forme d'une liste de données. Pour chaque nouvelle fenêtre créée, le gestionnaire placera sa structure de donnée dans la liste et l'associera à la tâche dont elle est originaire.

La fenêtre sera créée selon un style particulier. Le sous-ensemble des styles repris au tableau 6.2., montre la hiérarchie selon laquelle sont organisées les fenêtres d'une tâche.

Style visuel	Propriétaire spécifié?	Vie liée à ...	Relations visuelles
WS_OVERLAPPED	Non	Tâche	Superpose d'autres fenêtres et a toujours un titre et un cadre
WS_OVERLAPPED	Oui	Propriétaire	superpose le propriétaire, a toujours un titre et un cadre et est non visible si le propriétaire n'est pas visible
WS_POPUP	Non	Tâche	Superpose d'autres fenêtres
WS_POPUP	Oui	Propriétaire	Superpose le propriétaire et non visible si le propriétaire n'est pas visible
WS_CHILD	Oui	Propriétaire (parent)	Attaché au parent

Tableau 6.2. : Quelques styles visuels.

Notons encore que, chaque fenêtre sera identifiée par un descripteur. Un ***descripteur*** est une valeur renvoyée à l'application lorsqu'elle demande la création d'un ***objet Windows*** tel un bitmap, une fenêtre, un pinceau, une police de caractères, etc..

Les descripteurs sont également une caractéristique propre à Windows. En effet, Windows ne permet pas d'accéder directement aux structures de données des objets qui se trouvent en mémoire. Pour chaque objet créé, le programme reçoit un descripteur qui identifie cet objet. C'est donc par le biais de ce descripteur et des fonctions de l'API de Windows, qu'une application peut accéder aux données en mémoire.

C. Le gestionnaire de la mémoire

Le gestionnaire de la mémoire contrôle toute la mémoire disponible du système. Il alloue et libère dynamiquement les blocs et segments de mémoires nécessaires à une correcte gestion des applications. Il autorise également le programmeur à effectuer ces opérations par le biais des fonctions de l'API pouvant être appelées dans des programmes Windows.

Le gestionnaire organise la mémoire en deux tas : le tas global et le tas local.

Le ***tas global*** contient toute la mémoire disponible et est sujet à une allocation segment par segment.

Le ***tas local***, lui, est constitué de la mémoire occupée par le segment de données par défaut dont nous avons parlé ci-dessus.

6.1.2. Les applications Windows

Nous l'avons vu au point A. de la section précédente, Windows communique avec les applications par le biais de messages, qui sont en fait des appels de fonctions avec des paramètres qui apportent des précisions au programme appelé, l'événement venant de se produire.

Pour traiter et communiquer avec Windows, il faut que l'application dispose d'au moins une fenêtre et donc d'une fonction fenêtre exportée. De plus, pour qu'un programme Windows puisse effectuer correctement un certain traitement, il doit nécessairement avoir une certaine structure. Celle-ci a généralement deux grandes composantes : la ***fonction WinMain*** et les ***fonctions fenêtres***.

A. La fonction WinMain

WinMain est la fonction qui constitue le principal point d'entrée par lequel Windows pourra communiquer avec l'application. Cette fonction doit obligatoirement faire partie de l'application, bien qu'elle ne doive pas être exportée.

Ce qu'effectue généralement en premier lieu cette fonction, c'est la définition de la classe de la fenêtre principale de l'application. Lors de la définition de cette classe, une fonction fenêtre qui traitera tous les messages à destination des instances de cette classe est assignée. Une définition précise d'une classe de fenêtre pourra être trouvée au point A. de la section 6.1.3. Notons encore que, c'est également au début du programme qu'est prise en compte la possibilité d'avoir plusieurs instances de la même application.

En effet, la classe ne sera prise en compte par Windows que lorsque la première instance de l'application s'exécutera. Lors d'exécutions successives, le programme vérifiera si une instance de l'application existe déjà et selon, créera ou non une classe de fenêtres. La classe mémorisée par Windows ne sera détruite que lorsqu'il n'y aura plus aucune instance de l'application.

Ensuite la seconde mais non moins importante mission que doit remplir WinMain concerne le traitement des messages.

WinMain contient une **boucle de messages** qui, comme nous pouvons le voir à la figure 6.3., va rechercher un message dans une file d'attente de type "First In First Out" et remplit une structure de donnée qui est la structure d'un message Windows. Si un message est trouvé dans cette file, il sera passé aux fonctions **TranslateMessage** et **DispatchMessage** qui s'assureront du renvoi du message à la bonne fonction fenêtre. Sinon, **GetMessage** permetttra de rendre la main au gestionnaire de tâches pour que d'autres opérations puissent s'effectuer. Cette boucle s'exécutera jusqu'à ce qu'un message de fin de tâche y parvienne.

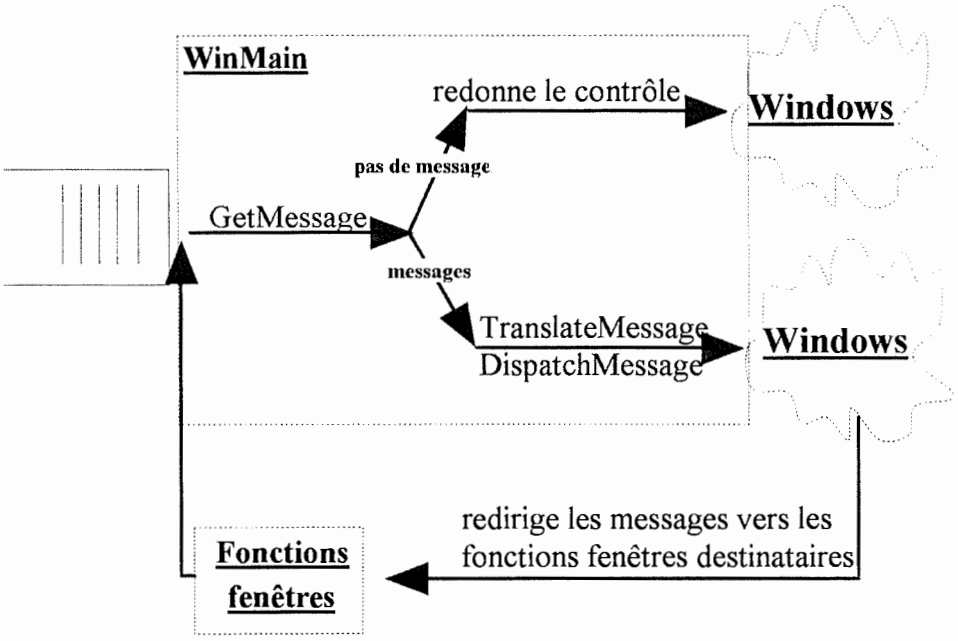


Figure 6.3. : Structure de la boucle des messages de WinMain

Néanmoins, comme nous l'avons vu précédemment, Windows peut ne pas passer par cette boucle de messages et expédier directement un message en faisant appel à une fonction fenêtre. Le fait que nous ayons deux méthodes pour prendre connaissance des messages réside dans le besoin de synchronisation entre Windows et les applications. Dans le cas où un message est "posté" dans une file d'attente, il doit attendre que le gestionnaire de tâches rende le contrôle à l'application pour pouvoir être traité. Ce traitement du message est immédiat dans le second cas et est souvent le chef de messages concernant la gestion de la fenêtre. La figure 6.4. montre de manière détaillée comment s'effectuent les interactions entre un programme Windows et le système.

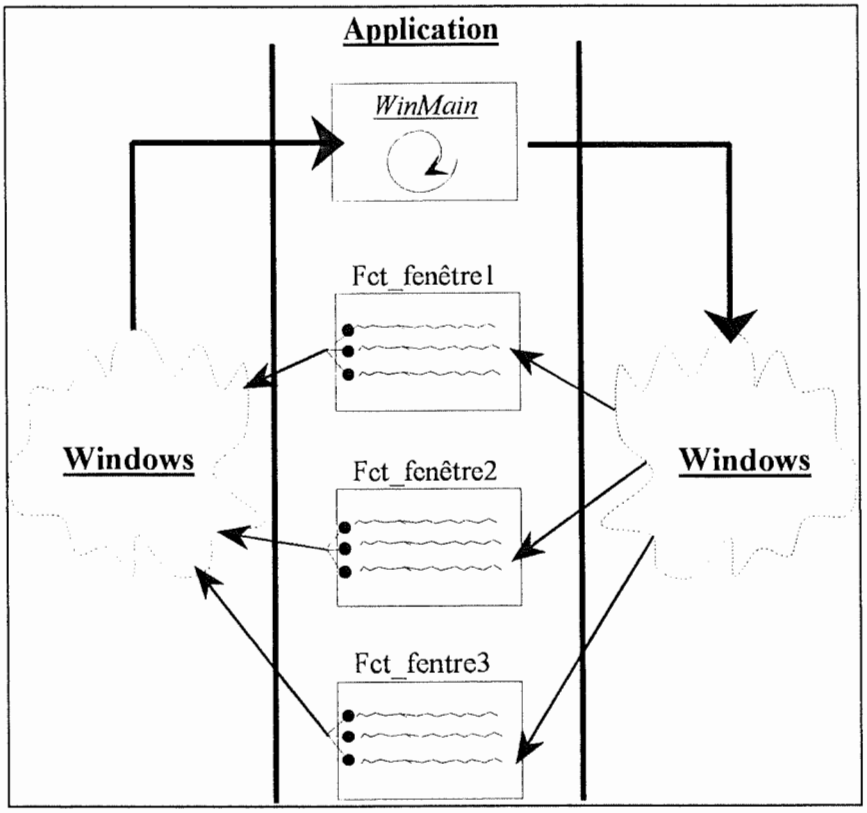


Figure 6.4. : Flux des appels entre une application Windows et le système

B. Les fonctions fenêtres.

Lorsqu'une fonction fenêtre veut traiter un message expédié par Windows, il faut qu'elle intercepte ce dernier. Pour ce faire, la fonction fenêtre utilise la commande "**switch**" qui lui permet de choisir, parmi l'ensemble des traitements qu'elle peut exécuter, le traitement qui correspond au message reçu. La fonction fenêtre est donc une suite de "**case**", interceptant et traitant un message bien précis.

Et les messages non traités?

A toute classe de fenêtres est associée un descripteur d'une fonction fenêtre qui traitera tous les messages à destination des instances de cette classe. Lorsque nous créons une fenêtre, une fonction fenêtre par défaut y est associée. Pour faire en sorte que ce soit notre fonction fenêtre qui traite les messages, nous utilisons la technique de subclassing.

Le ***subclassing*** consiste à remplacer le descripteur de la fonction fenêtre par défaut par le descripteur de notre fonction fenêtre. Comme nous connaissons le descripteur de la fonction fenêtre par défaut, nous pouvons lui passer tous les messages qui ne sont pas

traités par notre fonction fenêtre. La figure 6.5. montre la structure générale d'une fonction fenêtre.

```
long FAR PASCAL derouleur(descr_fen,msg,wParam,lParam)
        .
        .
 {
 switch (msg)
 {
 case WM_MOUSEMOVE
        ...
        break;
 case WM_LBUTTONDOWN
        ...
        break;
 ...
 }
CallWindowProc(descr_old_fct_fen,descr_fen,msg,wParam,
                              lParam);
 }
```

Figure 6.5. : structure générale d'une fonction fenêtre

Pour élaborer le traitement d'un message, il nous faut encore utiliser les fonctions de l'API de Windows.

6.1.3. Windows : une vision orientée objet

La conception de Windows a assez largement été influencée par les concepts orientés objets (O.O.).

Nous reprendrons dans cette section quelques mécanismes propres à l'orienté objet et montrerons que nous pouvons les retrouver dans Windows.

Cependant, à aucun moment nous n'affirmerons que Windows est un environnement de programmation O.O.. Nous sommes tout à fait conscients que, dans ce cadre, le système est limité. Par notre démarche, nous voulons simplement montrer que certains mécanismes propres, notamment, à la gestion des fenêtres ressemblent fortement à certains mécanismes de l'O.O..

A. Les objets et les classes d'objets

Nous l'avons vu, chaque fenêtre est composée d'une structure de donnée et d'une fonction fenêtre. La structure de donnée constitue l'ensemble des attributs d'un objet. La fonction fenêtre constituée des traitements que la fenêtre devra effectuer en réponse à

certains messages, correspond aux services offerts par l'objet durant sa vie. Nous pouvons déduire de ceci qu'une fenêtre ressemble très fort à un objet au sens O.O. du terme.

Windows permet également de définir une classe de fenêtres. Celle-ci, comme nous pouvons l'observer à figure 6.6., permet de définir toute une série d'attributs et de services qui seront génériques pour toute fenêtre créée à partir de cette classe.

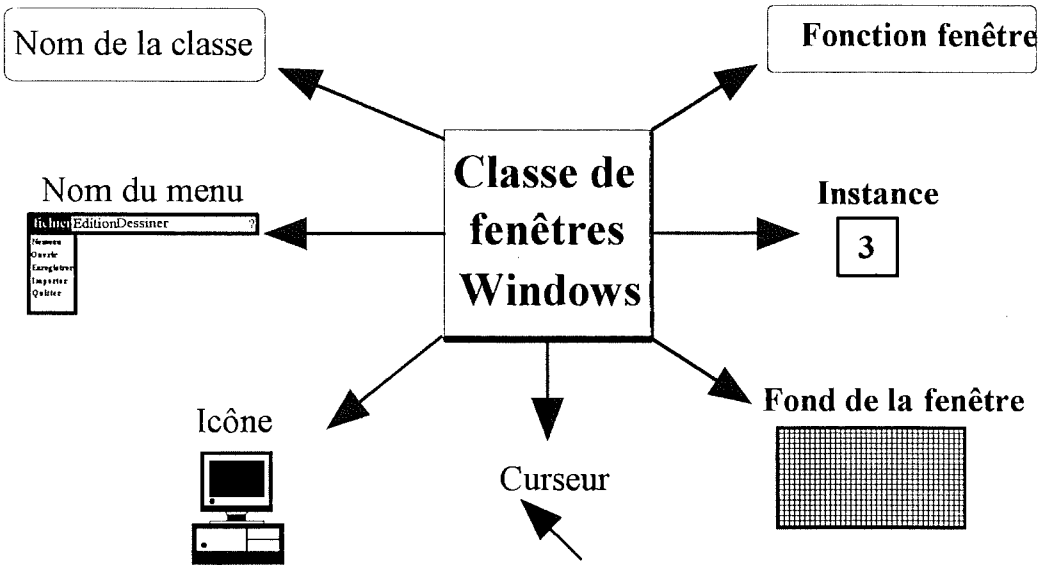


Figure 6.6. : Une classe de fenêtres Windows

Comme une instance d'une classe d'objet, une fenêtre aura les mêmes propriétés et comportements que sa classe mais possédera des valeurs particulières pour les attributs qui lui sont spécifiques.

Comme en O.O., les fenêtres communiquent par l'envoi de messages. Chaque message correspond, en fait, à un traitement, un service que l'objet devra exécuter.

B. Le masquage d'informations

Le masquage d'informations est le propre de la fonction fenêtre. Pour un certain nombre de messages reçus, un traitement sera effectué. L'objet qui déclenche le service, qui appelle en fait une fonction fenêtre avec certaines valeurs en paramètres, ne connaît rien de la façon dont est implémenté le service qu'il requiert.

C. L'overloading

L'overloading consiste à utiliser un même nom pour plusieurs réalisations différentes. Ce mécanisme est typique en Windows. Les messages expédiés aux fenêtres

sont souvent identiques. ce sont des messages tels WM_MOUSEMOVE, WM_LBUTTONDOWN, WM_KEYDOWN, ... dont la signification est explicite. Mais, le traitement correspondant à ces messages varie selon la fonction fenêtre qui est rattachée à une fenêtre et/ou selon la fenêtre elle-même.

D. L'héritage

L'héritage est "un mécanisme grâce auquel une classe définit les services et la structure de donnée de ses instances comme un sur-ensemble de la définition d'une ou de plusieurs autres classes" [Meye,88].

Ce mécanisme existe aussi, de manière éloignée, sous Windows. Il est en effet possible, sur base des informations d'une ou de plusieurs classes de fenêtres, de modifier certaines de leurs propiétes et comportements, et de conserver tout ce dont on veut hériter dans une nouvelle classe. Il suffit alors d'assigner un autre nom à cette classe et nous obtenons une sous-classe des classes initiales.

E. L'encapsulation

Rappelons-le, l'encapsulation est un mécanisme par lequel des connaissances et des services qui ont des points communs au niveau conceptuel sont rassemblés.

Nous trouvons également ce mécanisme dans Windows. Tous les services offerts par une fenêtre sont regroupés dans sa seule fonction fenêtre.

6.2. Les outils utilisés pour l'élaboration des exercices

6.2.1. Le compilateur Borland C++

Nous avons choisi d'utiliser le compilateur **Borland C++** .

En effet, grâce à divers travaux effectués à l'Institut, nous avions pu nous familiariser avec l'environnement de programmation du TurboPascal. Cet environnement est très proche de l'environnement du Borland C++. L'utilisation de ce logiciel nous évita donc de devoir apprendre la manipulation d'un nouvel outil.

Ce compilateur offrait, bien entendu, le support de la programmation Windows. Il offrait également divers outils qui nous aidèrent lors de l'élaboration des exercices. Il y avait notamment l'éditeur de ressources, le WhiteWater Resource Toolkit(WRT), dont nous avons parlé au chapitre 2. Cet éditeur nous a permis de mettre sous un fichier de ressources tous les dessins qui sont utilisés pour l'élaboration des exercices spécifiés lors du chapitre précédent.

Outre l'implémentation des applications, le Borland C++ permettait également l'élaboration de Bibliothèques à liens dynamiques dont nous avons abordé les principes également dans le chapitre 2.

6.2.2. Le logiciel de dessin DRAW+

Pour élaborer les divers dessins qui composent nos exercices et jalonnent notre mémoire, nous avons utilisé cet outil.

DRAW+ est un logiciel de dessin qui permet d'obtenir des graphiques sous format vectoriel d'une grande qualité. Malheureusement, Le WRT ne supporte pas ce genre de graphisme. Ce dernier utilise le format bitmap, littéralement une table de bits où un pixel est représenté par un ou plusieurs bits selon que l'on traite un graphique en noir et blanc ou en couleurs. Lorsque nous transférons un dessin du DRAW+ au WRT, la conversion entre le format vectoriel et le format bitmap entraîne une perte d'informations assez importante. Cela influe assez nettement sur la qualité du graphisme.

DRAW+ possède également une série assez impressionnante de graphiques déjà élaborés; son CLIPART.

6.3. Les problèmes rencontrés

Nous exposerons, dans cette section, les divers problèmes qui existent lorsqu'un programmeur s'essaie à la programmation d'applications Windows. Nous classerons ces problèmes selon l'ordre chronologique dans lequel nous les avons rencontrés.

6.3.1. La documentation

La documentation constitue un problème essentiel pour pouvoir programmer sous Windows.

Premièrement, les investissements en temps de lecture sont assez considérables. Le livre de Petzold [Petz,90] constitue le premier pas nécessaire pour connaître l'environnement, ses principes et son fonctionnement. Signalons, toutefois, que ce livre pourrait fort bien être suffisant pour élaborer de petites applications en se basant uniquement sur les objets et fonctionnalités déjà offertes par Windows.

Pour tenter d'expliquer cela, prenons l'exemple du dérouleur textuel à sélection directe. Il est le premier objet que nous ayons réalisé et est constitué d'objets graphiques offerts par Windows : deux boutons et une liste de sélection. Pour travailler avec des objets qui sont déjà implémentés par Windows, il faut un à deux mois d'apprentissage. Et s'il y a peu d'interactions entre les objets graphiques d'un programme, de petites applications peuvent être réalisées sans problème.

Outre la définition des objets, il faut encore contrôler tout le dialogue entre ces derniers. Par exemple, un click sur l'objet flèche de défilement qui est un bouton Windows, permet de déclencher un defilement des éléments de la liste de sélection. Il y a donc tout un flux de messages à contrôler.

De plus, le comportement par défaut des objet Windows peut ne pas se révéler pertinent pour les objectifs qui leurs sont assignés. Il faut alors, par la technique de **Subclassing**, créer une fonction fenêtre réalisant des services qui nous sont propres et remplacer la fonction fenêtre par défaut.

Si un niveau de programmation plus poussé est abordé, le livre de Petzold se révèle nettement insuffisant. N'abordant que de manière trop générale certains points, il ne montre de plus que des programmes simples avec peu ou pas de dialogue entre objets alors que c'est là que réside toute la difficulté à réaliser une bonne application Windows. Afin de pallier cette lacune, le livre de Wilton [Wilt,91], qui est un livre de programmation avancée, intéressera le lecteur averti surtout par l'expérience de l'auteur pour ce type de programmation. Cela rend l'ouvrage attrayant et fort utile.

Ensuite, viennent les documents de référence pour les fonctions de l'API de Windows. Nous regrettons que ces documents soient incomplets et contiennent un certain nombre d'erreurs. Nous avons trouvé plus d'une erreur dans la correspondance entre les paramètres des fonctions documentées et l'implémentation. Ce genre d'erreurs, pratiquement indétectables, nous a fait perdre des jours précieux. Nous espérons que les versions suivantes des documents ne contiendront plus ce genre d'erreurs.

Pour terminer, il faut également signaler l'impossibilité de trouver des documents concernant la programmation du graphisme. En effet, pratiquement tous nos objets sont issus de manipulations de bitmaps. Il est très difficile sans une documentation appropriée de

réaliser, par exemple, la lecture d'un fichier bitmap et de l'afficher dans une fenêtre. De plus, toutes les personnes contactées ayant déjà réalisé des applications Windows n'ont pu nous aider.

6.3.2. La programmation

La première difficulté consiste à comprendre correctement quelles sont les fonctions des divers fichiers qui vont permettre l'élaboration d'une application Windows. Il y a toujours trois fichiers de base plus éventuellement le fichier des ressources.

Les trois fichiers sont le fichier source, le fichier .DEF qui contient toute une série d'informations sur les caractéristiques du programme nécessaires à un chaînage correct et le fichier .MAK qui permet par l'exécution de la commande NMAKE de compiler et de chaîner l'application sans devoir faire appel à l'environnement de programmation du BORLAND C++. Ce dernier fichier n'est pas nécessaire si la commande NMAKE n'est pas utilisée.

Le fichier .DEF est très important. C'est là que doivent être signalées, notamment, les diverses fonctions fenêtres contenues dans l'application. Si l'on oublie de le faire, Windows ne peut pas communiquer correctement avec l'application.

Le problème suivant provient de la communication par messages. Le cheminement dans une application Windows est très difficile à suivre. Réaliser le dialogue entre objets graphiques ne l'est certainement pas moins. Si nous pouvions donner un conseil, il serait de baser la conception d'une application Windows sur la représentation graphique d'OBLOG. Elle a l'avantage d'être très parlante et de se rapprocher des interactions entre objets que nous avons rencontrées dans le cadre de la programmation.

Un autre problème, qui a trait à la communication par messages, est la quantité de données que peuvent se transmettre deux objets graphiques. Il est impossible en Windows de transmettre par message plus de 48 bits de données. A regret, nous n'avons trouvé dans les divers documents consultés aucune trace d'information nous signalant cette énorme lacune de l'environnement. Ce n'est que lorsque le besoin de transférer une certaine quantité de données à un objet se fait sentir, que cette faiblesse est découverte. Le seul moyen de pallier cette carence consiste à enregistrer les données sous forme de fichier. Il suffit alors, après enregistrement, de signaler à l'objet qui doit recevoir les données, qu'il peut en commencer la lecture et le traitement.

Mais, la difficulté la plus importante provient certainement du débogage. Il n'est pas possible d'effectuer l'affichage d'une suite de valeurs pour examiner s'il y a un problème et savoir où se situe ce problème, comme il est couramment fait usage sous MS-DOS. Il est de plus assez lourd de faire afficher des fenêtres avec un certain message pour ce genre de détection. Une solution que nous avons trouvée, sans affirmer que ce soit la meilleure, consiste à élaborer une fonction "trace" qui effectue l'écriture d'un string ou d'une valeur numérique dans un fichier texte. Le programmeur utilise donc cette fonction pour suivre le cheminement de l'application tout en lui permettant d'examiner les valeurs de diverses variables. Le programmeur peut alors, suite à l'éxécution de l'application, examiner le contenu du fichier et repérer d'où provient la faute.

Chapitre 7 : Vers un logiciel d'élaboration d'interfaces

Introduction

Comme il est indiqué tout au long de ce travail, notre tâche est d'offrir un ensemble d'outils permettant à un informaticien de concevoir des applications interactives pour personnes handicapées. Cependant, comme nous l'avons déjà signalé, un informaticien n'est pas un spécialiste dans le domaine du handicap. Il éprouvera donc des difficultés à concevoir une interface parfaitement adaptée à ce type de population. En conséquence, la démarche est à poursuivre pour arriver à un logiciel utilisable par des éducateurs. Connaissant mieux que quiconque l'utilisateur, l'éducateur sera un meilleur intermédiaire que l'informaticien, à condition qu'il puisse disposer d'un logiciel qui lui soit adapté.

Nous allons dans le présent chapitre exposer les jalons nécessaires à la réalisation d'un tel logiciel. Ces derniers sont des idées sur ce que sera le logiciel, sur ce dont il serait bien de disposer. Celles-ci ont été déterminées au retour du stage en Angleterre, lorsque nous pensions réaliser ce type de logiciel. Bien que ne constituant pas une base formelle et précise, elles pourront aider les étudiants qui continueront le travail.

7.1. Eléments constituant le logiciel

Comme nous l'avons indiqué dans le chapitre 2, le logiciel regroupera une boîte à outils qui sera le principal support du logiciel. De plus, il nécessitera un composant de paramétrage du dialogue entre l'ordinateur et la personne handicapée, un éditeur de dialogue, un composant de vérifications syntaxiques et sémantiques et un interpréteur de dialogue. Il sera donc un "Système de Développement d'Interfaces Usagers" (SDIU) au sens où nous l'avons vu au chapitre 2. Une architecture générale de ce dernier est donnée à la figure 7.1.

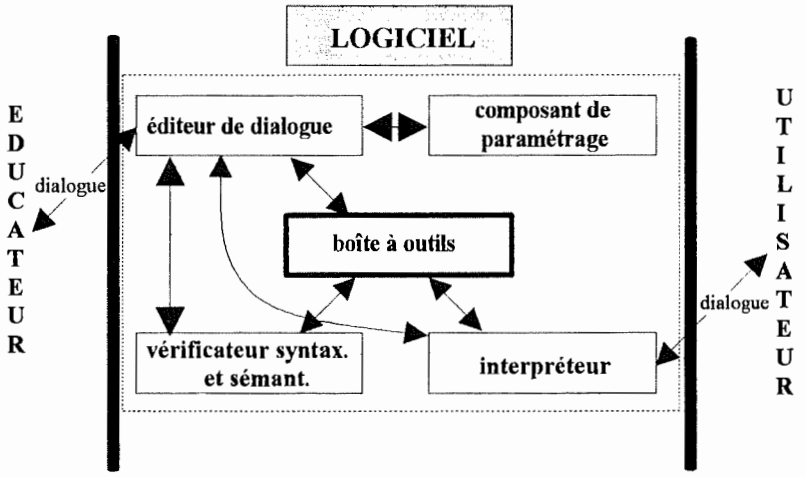


Figure 7.1: L'architecture

Il devra, en outre, présenter une interface aisément compréhensible par les éducateurs. Une telle interface présentera sous forme de menus les différents outils parmi lesquels l'éducateur pourra puiser afin de construire son interface.

Il sera également nécessaire de prévoir une aide à l'utilisation du logiciel.

Nous allons à présent donner quelques indications sur les différents éléments du SDIU.

7.1.1. Les outils

A. Les objets graphiques

Nous avons indiqué dans le chapitre 4 que nous pensons que l'ensemble des outils graphiques (objets) réalisés est suffisant. Nous avons cependant affirmé que pour en être sûrs, il faudra procéder à l'évaluation de ces derniers. Une telle évaluation permettra, si l'ensemble des objets offerts est insuffisant, de dégager d'autres objets graphiques à offrir.

B. Les actions

En plus des objets graphiques, il sera nécessaire de travailler les actions. Ces dernières sont les conséquences des interactions entre l'interface décrite par l'éducateur et la personne handicapée. Il faudra, comme nous l'avons fait pour les objets graphiques, analyser et implémenter les actions sur les objets.

Un exemple d'action est le déplacement d'objet vers une destination. Dans ce cas, lorsque la personne handicapée sélectionnera l'objet initiateur, l'action sera déclenchée, c'est-à-dire que l'objet sera déplacé vers la destination indiquée.

La création de sons sera une autre action à incorporer dans un prochain travail. Elle sera décrite comme une action, car elle est bien une conséquence des interactions avec l'utilisateur. Notons, par exemple, que dans le logiciel de simulation d'un jeu de marionnettes [Brou, 90] le son est une conséquence du click de l'utilisateur au moyen de la souris sur une icône illustrée par le dessin d'une oreille.

Certaines actions sont déjà implémentées. Il s'agit bien évidemment de la création et de la destruction d'objets graphiques.

Comme nous l'avons vu dans les chapitres 2 et 6, la structure événementielle de Windows permet, pour toute fonction fenêtre d'une application, d'enrichir ses fonctionnalités par l'interception et le traitement de nouveaux messages. C'est de cette façon qu'il faudra implémenter les nouvelles actions.

C. Les variables, les opérations et les traitements élémentaires

Un autre outil qu'il serait intéressant d'offrir permettra d'associer des variables à des objets graphiques. En effet, nous nous sommes aperçus que les dialogues entre l'interface et la personne handicapée nécessitent souvent de petites opérations élémentaires. Celles-ci peuvent être considérées comme des évaluations d'expressions. Par exemple, la comparaison de deux nombres, des opérations arithmétiques sur des expressions, etc. De plus , il serait également intéressant de disposer de petits traitements de type "if...then...else" ou "while...do...". Il faudra permettre aux éducateurs de réaliser ce type de traitements via des variables assignées dynamiquement et un ensemble d'opérations et de traitements prédéfinis.

Pour illustrer la pertinence de cet outil, il suffit de se souvenir du logiciel d'aide à la gestion des achats alimentaires [Tope, 91]. Dans ce dernier, l'utilisateur peut déterminer le nombre de personnes prenant part au repas grâce à un dérouleur. Une fois l'information recueillie, il est nécessaire, pour préparer le repas, de multiplier les quantités d'ingrédients par personne, par le nombre de personnes conviées au repas. Pour pouvoir réaliser ce dialogue, il faudra donc offrir la multiplication et la possibilité d'associer une variable à l'objet dérouleur.

Le lecteur intéressé trouvera la description d'un sous-ensemble d'actions, d'opérations et de traitements élémentaires à l'annexe 3.

7.1.2.Le composant de paramétrage

La conception d'une interface destinée à des personnes handicapées demande une étude approfondie. Le plus grand soin doit être apporté à l'étude des possibilités physiques et intellectuelles des utilisateurs. Nous avons vu, au chapitre 1, que la population des utilisateurs était hétérogène. Il est donc essentiel de prévoir une interface adaptable.

L'adaptation au niveau du contenu de l'interface sera réalisée par l'éducateur lorsqu'il décrira les objets et les actions associées aux objets. L'éducateur devra, par

exemple, prendre soin de choisir des libellés et des pictogrammes pertinents pour l'utilisateur, il devra donner aux objets des tailles et des couleurs significatives, il devra choisir les actions à associer aux objets pour que la tâche de l'utilisateur face à l'exercice apparaisse clairement, ...

Il ne faut cependant pas perdre de vue que les problèmes cognitifs des utilisateurs peuvent se traduire au niveau moteur par des problèmes de coordination de mouvements. Ces utilisateurs seront d'ailleurs parfois incapables d'utiliser la souris. Nous voyons donc que, outre l'adaptation du contenu de l'interface, il faudra permettre à l'éducateur d'adapter ses exercices en fonction des capacités de l'utilisateur à manipuler une interface graphique par différents moyens d'interaction. Cette adaptation sera réalisée au moyen d'un outil de paramétrage.

Il existe de nombreux moyens d'interaction :

La souris est indiscutablement le moyen d'interaction le plus souvent utilisé pour la manipulation directe. Elle permet des déplacements du curseur et autorise un (ou plusieurs) mécanisme(s) de sélection. Notons cependant que l'utilisation de la souris requiert une certaine précision dans le geste et une certaine capacité d'abstraction pour en comprendre le mécanisme. Elle ne sera donc pas utilisable par tous les utilisateurs.

Le suiveur (en anglais, tackball) permet les mêmes opérations que la souris mais ne doit pas être déplacé comme cette dernière. En effet, les mouvements du curseur sont obtenus en faisant rouler la boule de contrôle avec la main. Le suiveur est une aide précieuse pour l'utilisateur ne pouvant réaliser que de légers mouvements avec la main.

L'écran tactile et le **crayon optique** permettent à l'utilisateur de désigner un point directement sur l'écran. Ils sont donc particulièrement adaptés aux utilisateurs n'ayant qu'une faible capacité d'abstraction. L'écran tactile est plus simple à manier que le crayon optique car il n'y a pas d'intermédiaire entre l'écran et l'utilisateur. Il est cependant moins précis que ce dernier à cause de l'épaisseur d'un doigt. Ces deux moyens d'interaction provoquent une certaine fatigue du bras de l'utilisateur.

Le joystick ne demande que des mouvements de faible amplitude et permet des changements de directions aisés quoique lents et peu précis. Il existe différents types de joysticks. Certains ont un seul bouton de commande permettant la sélection, d'autres en ont plusieurs, certains sont manoeuvrables à l'aide de quelques doigts, d'autres à une main, ...

La technologie **des systèmes de détection des mouvements oculaires** et de **reconnaissance de la parole** offre un très grand potentiel mais est encore fort coûteuse et

pose quelques problèmes de précision. En outre, les méthodes de sélection ne sont pas toujours satisfaisantes et occasionnent parfois des sélections accidentelles.

Ces différents moyens d'interaction sont les plus indiqués pour la manipulation directe. Ils sont appelés ***les périphériques de désignation*** (en anglais, pointing devices). "Ils contrôlent un pointeur qui peut être déplacé vers n'imporque quelle position de l'écran par n'importe quelle route choisie"[Colv, 90]. Ils permettent donc à l'utilisateur de désigner et de sélectionner directement un objet à l'écran ou de positionner un curseur sur l'objet de façon relativement aisée avant de le sélectionner. Les périphériques permettant de désigner et de sélectionner directement un objet à l'écran sont ***les périphériques de désignation absolus***. Ce sont les écrans tactiles, les crayons optiques, les systèmes de reconnaissance de la parole et les systèmes de détection des mouvements oculaires. Les périphériques permettant de positionner un curseur sur l'objet de façon relativement aisée avant de le sélectionner sont ***les périphériques de désignation relatifs*** tels la souris, le suiveur et le joystick.

Cependant, d'autres moyens d'interaction sont nécessaires pour les personnes présentant des déficiences motrices tellement importantes que les mouvements réalisables sont extrêmement limités et imprécis; ce sont les **interrupteurs**.

Il existe un nombre impressionnant d'interrupteurs. Ces derniers peuvent être simples ou multiples. Les **interrupteurs multiples** permettent le déplacement à l'écran et la sélection alors que les **interrupteurs simples** ne permettent que la sélection.

L'outil de paramétrage permettra à l'éducateur de paramétrer l'interface en fonction du moyen d'interaction qui sera utilisé.

Quel que soit le périphérique utilisé, il faudra filtrer ses entrées pour éviter les sélections accidentelles ou les répétitions automatiques par la personne handicapée. Différents paramètres permettent d'effectuer le filtrage.

Les personnes handicapées peuvent présenter des tremblements ou d'autres troubles causes de sélections accidentelles de courte durée. Un paramètre, appelé l'***"Input acceptance time"*** permettra de les rejeter. Grâce à lui, l'éducateur pourra spécifier le temps minimum pendant lequel il faudra que l'utilisateur tienne le moyen d'interaction dans la position correspondant à la sélection pour que celle-ci soit prise en compte par le système. Ce temps peut être utilisé pour ne pas prendre en compte une mauvaise sélection de l'utilisateur. En effet, ce paramètre permettra à l'utilisateur d'effectuer une nouvelle sélection avant que la première ne soit prise en compte.

De la même façon, le ***"Post acceptance delay"*** permettra à l'éducateur de spécifier le laps de temps qui doit s'écouler après une sélection, avant qu'une nouvelle sélection ne puisse être prise en compte.

D'autres paramètres de filtrage des saisies par les moyens d'interaction peuvent être trouvés dans le document de D. Colven de l'ACE Centre [Colv, 90]. Il s'agit du ***"Debounce time"***, de l'***"Edge triggering"*** et des ***"Repeat functions"***.

Ces paramètres sont importants. Ils permettent, en effet, de franchir une étape supplémentaire dans l'adaptation de l'application à l'utilisateur. Des choix inopportuns concernant les valeurs des différents paramètres conduiront à une réduction de la qualité de la communication.

Outre le filtrage, il faudra effectuer des modifications au niveau de la présentation de l'interface en fonction du moyen d'interaction utilisé. Donnons-en quelques exemples.

Si le moyen d'interaction utilisé est un interrupteur multiple, il serait lourd, pour l'utilisateur, de déplacer le curseur de quelques pixels à la fois jusqu'à l'objet qu'il veut sélectionner. Nous savons, en outre, que la position qu'il essayera d'atteindre correspond à celle d'un des objets graphiques de l'interface sélectionnable à ce moment du dialogue.

Si l'interrupteur est simple, l'utilisateur ne pourra effectuer aucun déplacement.

En conséquence, quel que soit le type d'interrupteur utilisé, il faudra modifier la présentation de l'interface pour permettre à l'utilisateur de visionner les divers choix qu'il peut effectuer. Pour ce faire, il faudra réaliser une mise en évidence successive des différents objets graphiques sélectionnables; c'est ***"le balayage"*** (en anglais, scanning). L'objet prêt à être sélectionné par un interrupteur est celui qui est mis en évidence.

Il convient de donner à l'éducateur la possibilité de déterminer les objets à balayer à un moment donné, dans un écran donné. Ils pourront être les objets présents dans une zone de l'écran ou un sous-ensemble de ces derniers. Le balayage sera exécuté jusqu'à ce que l'utilisateur ait opéré une sélection.

Le balayage devra être automatique lorsque l'interrupteur est simple puisque ce type d'interrupteur ne permet pas de déplacement. Dans ce cas, en plus des objets à balayer, l'éducateur devra pouvoir imposer un ***temps de balayage*** (en anglais, scanning time). Ce dernier représente le temps pendant lequel l'objet est mis en évidence pour indiquer qu'il est prêt à être sélectionné.

Lorsque l'interrupteur est multiple, l'ordre dans lequel les objets seront balayés sera déterminé par l'utilisateur.

Les interrupteurs ne permettront pas l'utilisation du dérouleur puisque ce dernier est vu comme un objet unique. Il ne sera, dès lors, pas possible d'effectuer le balayage des différents éléments le constituant, ce qui réduit à néant l'utilité de cet objet.

Les données relatives au paramétrage devront être stockées pour pouvoir être exploitées par l'interpréteur décrit en 7.1.5.. Un schéma illustrant cette affirmation est donné à la figure 7.2..

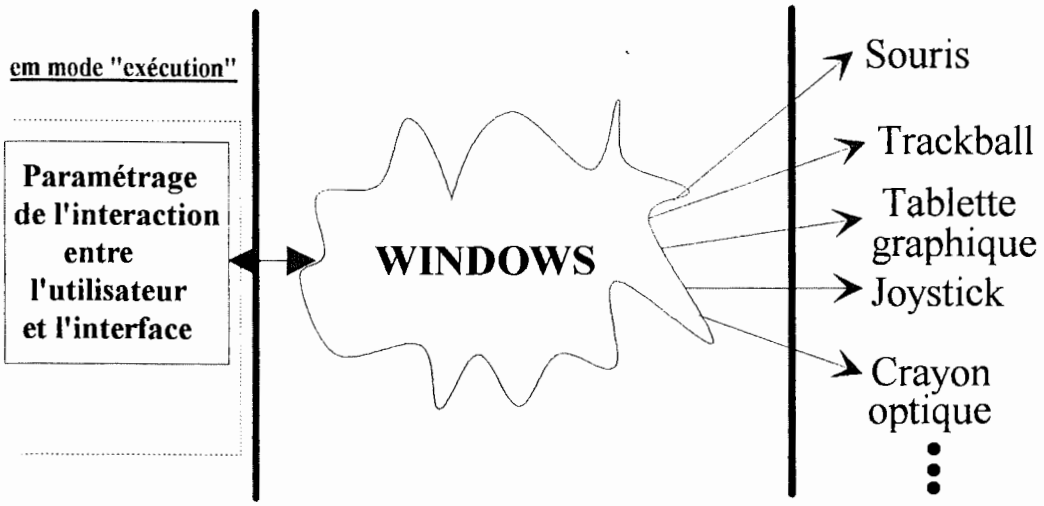


Figure 7.2 : Le paramétrage

7.1.3.L'éditeur de dialogue

Cet éditeur doit permettre à l'éducateur de décrire le dialogue de manière interactive. Il doit donc permettre de décrire les objets composant ce dernier, les variables associées aux objets, les actions à effectuer sur les objets, les opérations à effectuer sur les variables, les traitement élémentaires relatifs à ces dernières et les conditions de déclenchement des actions, des opérations et des traitements.

Pour décrire les objets et les variables nécessaires à l'élaboration d'un exercice, l'éducateur devra pouvoir, par le biais de menus, sélectionner l'objet interactif de son choix. Ensuite, le remplissage d'une forme devra lui permettre de définir toutes les caractéristiques de l'objet. Le placement de l'objet dans la zone désirée devra s'effectuer par le biais de la souris.

Les actions, les opérations et les traitements nécessiteront certaines informations. Citons, par exemple, une action qui serait le déplacement d'un objet. Une telle action sera décrite par l'identifiant de l'objet à déplacer et par la destination de ce dernier. En particulier, les informations d'une action de création d'objet sont constituées de tous les éléments descriptifs de l'objet et des variables à lui associer. Une opération de comparaison de type "<=" nécessitera deux informations permettant d'identifier les deux expressions à comparer. Un traitement de type "if...then...else..." nécessitera des informations qui sont des opérations. L'opération située après le "if" sera une opération à résultat booléen.

L'éducateur devra, de plus, pouvoir déterminer des suites d'actions, d'opérations ou/et de traitements à effectuer. Chaque élément de la suite aura une ou plusieurs cause(s) de déclenchement. Les causes de déclenchement seront :

- d'autres éléments de la suite. Par exemple, l'action X déclenche l'opération Y,

- une ou plusieurs action(s) de l'utilisateur. Par exemple, la sélection d'un ou de plusieurs objet(s) grâce à un moyen d'interaction déclenche une action.

Toutes les données relatives à la description du dialogue devront être examinées par le composant de vérifications syntaxiques et sémantiques présenté en 7.1.4.. Puis elles devront être sauvegardées selon un format garantissant l'absence de perte d'informations pour être analysées par l'interpréteur de dialogue. Cette structure pourrait être une table, appelée **table de correspondance**, de la forme suivante :

causes de déclenchement	type d'action, d'opération ou de traitement	paramètres	suivant

avec

causes de déclenchement : identifiant d'un ou de plusieurs objets à sélectionner ou rien si l'action, l'opération ou le traitement n'est pas déclenché par une sélection.

suivant : numéro de l'entrée de la table correspondant à l'élément suivant à déclencher, s'il y en a un. 0 si l'élément n'en déclenche pas d'autres.

La façon de présenter cet éditeur de dialogue à l'éducateur constituera une partie de l'interface du SDIU.

7.1.4. Le composant de vérifications syntaxiques et sémantiques

Ce composant sera chargé de vérifier les différentes données introduites par l'éducateur en vue du stockage de ces dernières. Il est important que les données stockées soient correctes sous peine d'entrer dans des situations incohérentes et incontrôlables lors de l'interprétation du dialogue décrit. La validité des données est essentiellement syntaxique. En effet, il n'est pas en notre pouvoir de contrôler que le dialogue fait bien ce que l'éducateur veut qu'il fasse. Tout au plus pouvons-nous contrôler la syntaxe et empêcher les oublis.

Voici quelques exemples de contrôles à effectuer :

- toutes les informations relatives aux différentes actions, opérations et traitements décrits ont été données par l'éducateur,

- ces informations sont syntaxiquement correctes,

- chaque action, opération ou traitement sera exécuté à un moment ou à un autre durant le dialogue. C'est-à-dire que chacun de ces éléments est déclenché par la sélection d'un objet ou est indiqué comme étant le suivant d'un autre élément.

Quelques rares exemples de contrôles sémantiques pourront cependant être effectués. Citons le contrôle des propriétés d'icône et le contrôle du choix des objets lorsque le moyen d'interaction est l'interrupteur. En effet, les icônes insélectionnables et invisibles ou insélectionnables et amovibles seront systématiquement rejetées et il sera interdit à l'éducateur d'utiliser un objet dérouleur comme composant d'une interface sur laquelle l'utilisateur ne pourra agir que via un interrupteur. Les raisons de ces différents refus ayant été discutées dans les chapitres 4 et 7, nous ne nous y attarderons pas davantage.

7.1.5. L'interpréteur

Ce composant sera chargé de l'exécution du dialogue. Après la description de ce dernier, la détermination des paramètres de l'interaction par l'éducateur et après que le composant de vérifications ait analysé toutes les données, celles-ci seront prêtes à être interprétées par ce dernier composant.

L'interpréteur sera donc chargé de réaliser les différentes actions, opérations et traitements décrits lorsque les conditions de déclenchement de ces derniers, qu'il trouvera

dans la table de correspondance, seront réalisées. Il réalise donc le dialogue entre l'application et la personne handicapée. Nous pouvons le constater à la figure 7.3.

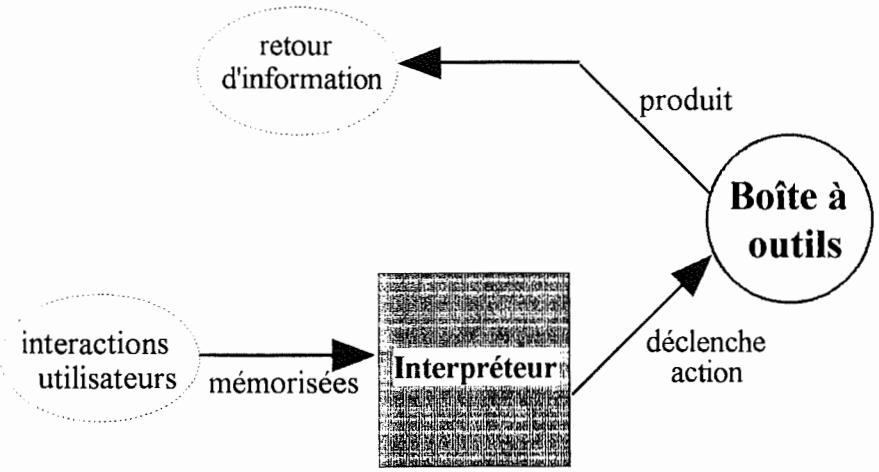


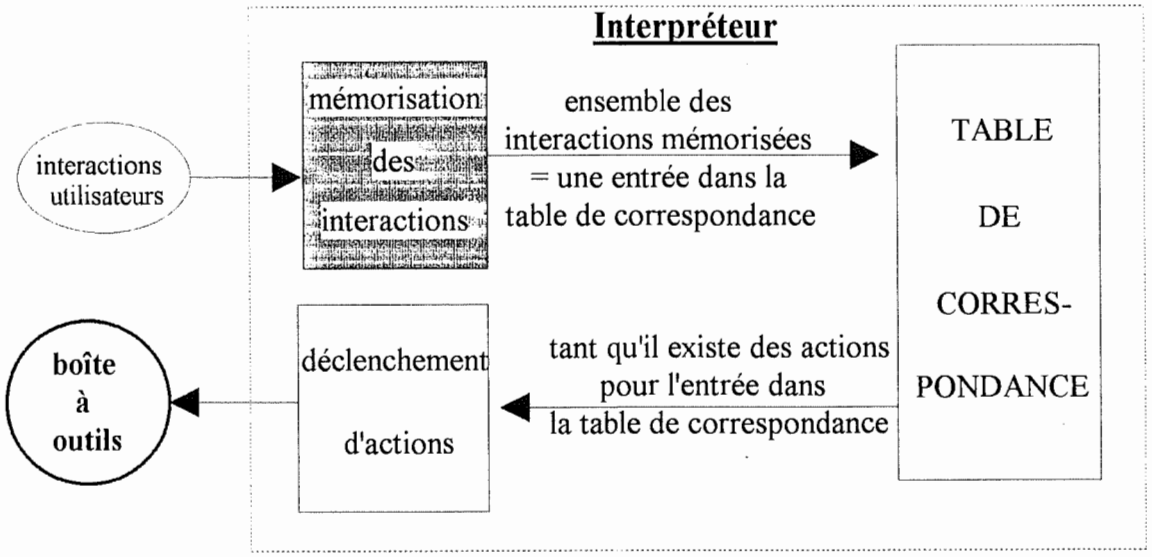
Figure 7.3. : Rôle de l'interpréteur

Nous avons vu, en 7.1.3., qu'une action, une opération ou un traitement pouvait être déclenché par la sélection d'un ou de plusieurs objets par l'utilisateur. Lorsque les causes de déclenchement d'un élément correspondent à un ensemble de sélections d'objets, il convient de mémoriser les identifiants des objets sélectionnés au fur et à mesure que l'utilisateur les sélectionne.

Illustrons cette affirmation par un exemple : un exercice nécessite la sélection de deux objets de la zone de travail. Lorsque ces derniers sont sélectionnés, le premier objet est déplacé vers le second et placé à côté de ce dernier. En outre, la zone de commande possède une icône sélectionnable permettant de terminer l'exercice. Si l'utilisateur sélectionne un des deux objets de la zone de travail puis l'icône de fin d'exercice, il faut que l'exercice se termine sans que le déplacement ne soit effectué. S'il sélectionne le premier objet de la zone de travail puis le second, il faut effectuer le déplacement. Si nous ne disposons pas d'un mécanisme de mémorisation des interactions précédentes, il sera impossible d'effectuer cette action.

En conséquence, l'interpréteur de dialogue exploitera les identifiants mémorisés comme suit : lors de la sélection d'un objet par l'utilisateur, il analysera la table de correspondance. S'il existe au moins une entrée pour laquelle la cause de déclenchement est l'identifiant de l'objet sélectionné, l'interpréteur écrasera l'ensemble des identifiants mémorisés et effectuera la séquence d'action, d'opération ou de traitement correspondant. Sinon, l'interpréteur mémorisera l'identifiant de l'objet sélectionné. S'il existe une entrée de la table de correspondance dont la cause de déclenchement est l'ensemble des identifiants mémorisés, l'action, l'opération ou le traitement élémentaire correspondant sera exécuté par

l'interpréteur. Il écrasera, de plus, l'ensemble des identifiants mémorisés. Une illustration du fonctionnement de l'interpréteur est donnée à la figure 7.4.



L'interpréteur sera, en outre, chargé d'exploiter les données relatives au paramétrage et donc de modifier l'interface selon le moyen d'interaction utilisé, comme nous l'avons expliqué au point 7.1.2..

7.1.6. L'interface du logiciel et l'aide à l'utilisateur

N'ayant pas pu, pour une question de temps, nous consacrer à l'étude approfondie de ce sujet, nous allons mentionner ici ce que doit présenter l'interface plutôt que d'essayer de répondre à la question du "comment le présenter".

En plus des possibilités classiques de gestion et d'édition de fichiers, l'interface du logiciel doit bien évidemment permettre à l'éducateur d'introduire de façon aisée toutes les caractéristiques décrites dans la section traitant de l'éditeur. Il s'agit de la description des objets et des actions de l'interface, de la description des variables, des opérations et des traitements élémentaires. Elle devra aussi lui permettre d'introduire les caractéristiques de paramétrage.

En outre, le logiciel pourra fonctionner selon deux modes : le **mode production** et le **mode exécution**. Le premier mode permettra à l'éducateur de décrire le dialogue alors que le second constituera l'exécution de ce dernier. L'interface du logiciel disparaîtra alors de l'écran pour céder la place au premier écran du dialogue décrit. C'est lorsque le logiciel présentera le premier écran que la personne handicapée deviendra l'utilisateur impliqué dans

la réalisation de la tâche. L'interface du logiciel devra donc aussi permettre à l'éducateur de passer d'un mode à l'autre lorsqu'il aura terminé la description du dialogue. Cela permettra d'implémenter et de corriger un exercice de manière interactive.

Comme dans tout logiciel, il faudra de plus fournir une aide à l'utilisateur. Dans notre cas, l'utilisateur sera l'éducateur. Il faudra l'aider à utiliser correctement le logiciel et à en comprendre l'interface.

Cette aide devra être la plus complète possible sous peine de voir les éducateurs se désintéresser totalement du logiciel. Elle pourra être fournie sous forme d'un menu ("help" classique) permettant de se diriger dans un ensemble d'explications classées en index. Elle pourra aussi être donnée sous forme de messages clairs, explicatifs d'erreurs réalisées. Ces erreurs seront détectées par le composant de vérifications syntaxiques et sémantiques. Les messages devront indiquer l'endroit de l'erreur, donner son diagnostic et afficher la forme de l'objet erroné ou la forme contenant la description de l'action erronée.

Conclusion

Il reste beaucoup de chemin à parcourir avant d'achever le projet global mais nous pensons avoir jeté les bases originales d'un logiciel de création d'interfaces en proposant un ensemble d'outils réutilisables destinés à faciliter la création d'interfaces pour personnes handicapées. Pour y parvenir, nous préconisons l'utilisation d'une méthode de conception orientée objet et la programmation sous Windows que nous avons utilisée.

BIBLIOGRAPHIE

Chapitre 1

[Boda, 90], F. Bodart, "Cours introductif aux interfaces homme-machine : notes de cours provisoires", FNDP Namur, 1989-1990.

[Boy, 91], G.A. Bay, Intelligent Assistant Systems, T.J. Press Limited, G.B., 1991.

[Card, 83], S.K. Card, T.P. Moran et A. Newell, the psychology of human-computer interaction, 1983.

[Cher, 90], F. Cheron et O. Berger, "Evaluation et amélioration d'un logiciel de gestion des comptes pour personnes handicapées mentales", FNDP, Namur, 1990.

[Hutc, 86], E.L. Hutchins, J.D. Hollan et D.A. Norman, "Direct manipulation interfaces" in User Centered System Design : New perspectives on Human-Computer interaction, 1986.

[Norm, 86], Donald A. Norman, "Cognitive Engineering" in User Centered System Design : New perspectives on Human-Computer interaction, 1986.

[Laur, 86], Brenda K. Laurel, "Interface as mimesis" in User Centered System Design : New perspectives on Human-Computer interaction, 1986.

[Shne, 87], Ben Shneiderman, Designing the User Interface : Strategies for Effective Human-Computer Interaction, 1987.

Chapitre 2

[Adle,91], Marc Adler, "An Overview of Windows 3.0 Development Tools" in Microsoft Systems Journal, juillet 91, pp 99-109.

[Ande,91], David A. Andersen, Bruce A. Sherwood, "Portability and the GUI" in Byte Magazine, novembre 91, pp 221-226.

[Bobr,86], D.G.Bobrow, S.Mittal and M.J.Stefik, "Experts systems: perils and promises" in Communication of the ACM, septembre 86, pp 880-894.

[Cass,90], Jean Cassagne, "Toolbook : l'Hypercard de Windows" in Science et Vie Micro, octobre 90, pp132-136.

[Eisn,90], Bill Eisner, "Extending off-the-shelf Windows Applications with Macros and DLLs" in Microsoft Systems Journal, novembre 90, pp 89-96.

[Eule,91], Laura Euler, Eric Maffei et Adam Rauch, "Create Real Windows Applications in a Graphical Environment by using Microsoft Visual Basic" in Microsoft Systems Journal, juillet 91, pp 57-69.

[Hart,89], Rex Hartson, "User-Interface Management Control and Communication" in IEEE Software, janvier 89, pp 62-70.

[Haye,89], Frank Hayes, Nick Baran, " A Guide to GUIs " in Byte Magazine, juillet 89, pp 250-257.

[Löwg,88], Jonas Löwgren, " History, state and future of User Interface Management Systems" in SIGCHI Bulletin.

[Myer,89], Brad A. Myers,"User-Interface Tools : Introduction and Survey" in IEEE Software, janvier 89, pp15-23.

[Petz,90], Charles Petzold, Programming Windows : the Microsoft Guide to Writing applications for Windows 3, Microsoft Press, 1990.

[Poto,88], Kathleen Potosnak, "Do icons make user interfaces easier to use?" in IEEE Software, mai 88, pp 97-99.

[Sena,90], Bernard Senach, "Evaluation ergonomique des interfaces homme-machine : une revue de la littérature" in Rapports de recherche INRIA, mars 90.

[Shaw,90], Richard Hale Shaw, 'Toolbook, an Interactive Development Environment for Windows" in Microsoft Systems Journal, septembre 90, pp 93-106.

[Shne,83], Ben Shneiderman, "DirectManipulation : A Step Beyond Programming Languages" in IEEE Computer, août 83, pp 57-69.

[Smit,82], D.C.Smith, C. Irby, R. Kimball, B. Verplanck, E. Harslem, "designing the Star User Interface" in Byte Magazine, Vol. n° 7, n° 4, avril 82, pp 242-282.

[Vald,89], Ray Valdés, "A Virtual Toolkit for Windows and the Mac" in Byte Magazine, mars 89, pp 209-216.

[Wilt,92], Richard Wilton, Microsoft Windows 3 : Outils de programmation avancée, éd. Microsoft Press, Dunod et PSI, 1992.

Références des divers systèmes mentionnés

- Algae

M.A.Flecchia and R.D.Bergeron, "Specifying Complex Dialogs in Algae", in Proceedings SIGCHI+GI 87, pp 229-234.

- Interactive Transaction System (ITS)

C. Wiecha, Bennett W., Boies S., Gould J., " Generating Highly Interactive User Interfaces" in CHI' 89, pp 277-282.

- Judgment-based Automatic Dialog Editor (JADE)

B. Vander Zanden, B. A. Myers, "Automatic, Look-and-feel Independant Dialog Creation for Graphical User Interfaces" in CHI' 90, pp 27-34.

- MacApp

K.J. Schmucker, "MacApp : An application framework", Byte Magazine, août 86, pp189-193.

-Open Dialogue

Appolo Computer, inc. Chelmsford, Massachussets.

- Peridot

B.A. Myers," Creating intercation Techniques by Demonstration", IEEE Computer Graphics and Applications, septembre 97, pp51-60.

- Rapid/USE

A.I.Waserman and D.T. Shewmake, "Rapid Prototyping of Interactive Information Systems" in SIGSoft Software Eng. Notes, decembre 82, pp 171-180.

- User Interface Development Environment (UIDE)

D. De Baar, J.D. Foley, K.E. Mullet, "Coupling application Design", CHI' 92, pp 259-266.

chapitre 3

[Cout, 91], L. Bass et J. Coutaz, Developping Software for the User Interface, Addison-Wesley Publishing Company, Inc, 1991.

[Cox, 86], Brad J. Cox, Object Oriented programming, Productivity products international, Inc, S. Hook, 1986.

[Booc, 86], G. Booch, Object Oriented development, IEEE Transactions on Software Engineering, 1986.

[Khos, 90], S. Khoshafian et R. Abnous, Object Orientation : Concepts, Languages, Databases, User Interfaces, Wiley, New York, 1990.

[Meye, 88], B. Meyer, Object Oriented Software Construction, Prentice-Hall International, UK, 1988.

[Parn, 84], D.Parnas, Software engineering principles, SNSL, 1984.

[Zeip, 92], J-M Zeippen et P.Dols, "Pragmatic introduction to the OBLOG Approach in System Design", Séminaire présenté dans le cadre du cours de Méthodologie de développement de logiciels, matières approfondies, ESDI, Lisbon, 1992.

chapitre 4

[Brou, 90], L. Brousmiche et X. Gillet, "Logiciel de simulation d'un jeu de marionnettes pour enfants infirmes moteurs cérébraux", FNDP, 1990.

[Cher, 90] : voir chapitre 1.

[Font, 91], A. Fontesse et B. L'homme, "Connaissance du corps interne : réalisation d'un logiciel d'apprentissage pour personnes ayant une déficience mentale", FNDP, Namur, 1991.

[Lepo, 90], T. Lepoutre et J-M. Roquet, "La personne handicapée mentale et la connaissance du corps humain : un logiciel d'apprentissage", FNDP, Namur, 1990.

[Prov, 91] : idem chapitre 2.

[Tope, 91], L. Topet et X. Vonèche, "L'autonomie de la personne ayant une déficience mentale : un logiciel d'aide à la gestion des achats alimentaires", FNDP, Namur, 1990.

[Warn,88], G. Warnant, "Eléments de modélisation du dialogue d'une application interactive, notes provisoires" in Cours d'interface homme-machine de F. Bodart, FNDP, Namur, 1988.

Chapitre 5

[Lepo, 90] : voir chapitre 4.

[Tope, 91] : voir chapitre 4.

Chapitre 6

[Meye, 88] : voir chapitre 3.

[Petz, 90] : voir chapitre 2.

[Wilt, 92] : voir chapitre 2.

Chapitre 7

[Brou, 90] : voir chapitre 4.

[Colv, 90], D. Colven, A common terminology for switch controlled software, ACE Centre, Oxford, 1990.

[Tope, 91] : voir chapitre 4.

Année Académique 1991-1992

"Elaboration d'une boîte à outils d'aide à la réalisation d'interfaces pour personnes handicapées" (ANNEXES)

Stéphanie Baudrenghien

Rudy Demo

Promoteur : **Mme Monique Noirhomme-Fraiture**

Mémoire présenté en vue de
l'obtention du grade de Licencié
et Maître en informatique

ANNEXE 1

PROJETS POUR PROGRAMMES EN INFORMATIQUE.

I. COLORIAGE.

A. COULEURS UNIES.

Progression : 1) a. - dessin simple colorié complètement.
- à côté : même dessin avec couleur amortie.
- l'élève doit sélectionner au moyen de la souris la même couleur parmi 2 choix.

b. même exercice mais on augmente le nombre de choix.

c. même exercice mais on varie la place de la couleur.

2) - même dessin colorié complètement.
- à côté : même dessin non colorié.
Progression : voir 1)

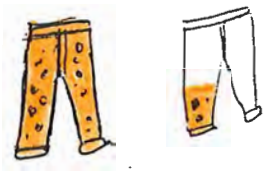
3) - même dessin non colorié.
Progression d'exercices : voir 1)

Variantes : - dessins.
- couleurs.

B. IMPRIME

Dessins : motifs sur fond uni.

Progression : voir A.1)



C. CONTRASTES FONCE-CLAIR

- dessin avec couleur foncée : ex. : rouge foncé.
- à côté même dessin avec la couleur claire correspondante amortie.

progression : voir A.1)

II. COMPREHENSION

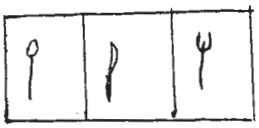
A. ASSOCIATIONS D'IDEES.

B. TROUVER L'INTRUS.

III. VOCABULAIRE.

CLASSEMENTS SEMANTIQUES.

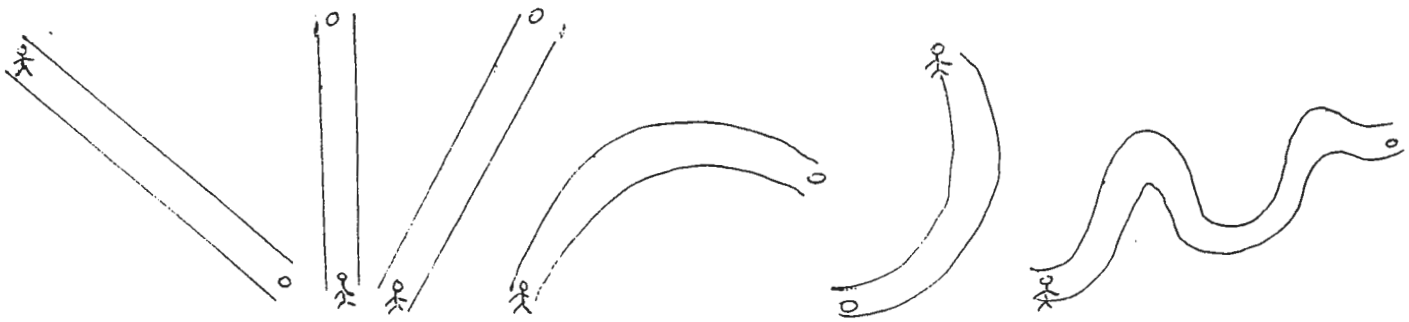
Exemple : différents ustensiles dessinés, l'élève doit les ranger dans un tiroir.



IV. PSYCHOMOTRICITE.

A. ORIENTATION SPATIALE.

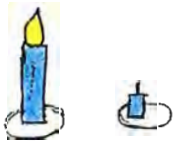
- Labyrinthes.



VARIANTES : - longueurs.
- largeurs.
- dessins.

B. ORGANISATION TEMPORELLE.

- Suites logiques.



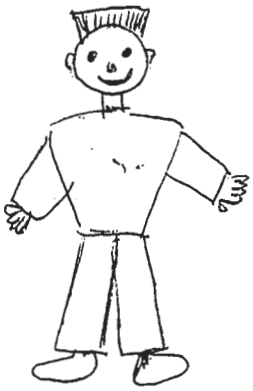
VARIANTE : nombre croissant d'images à remettre en ordre.

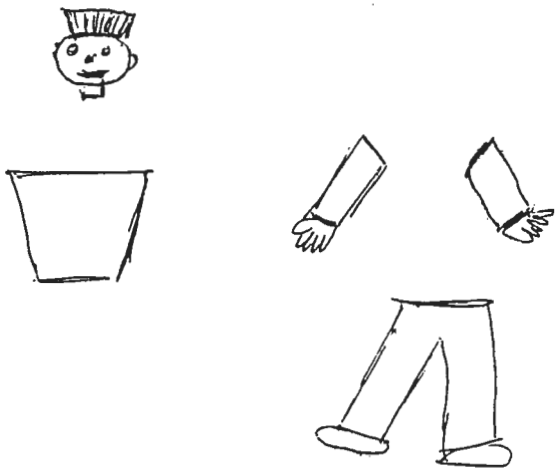
C. RYTHME.

- Séries à continuer.

D. SCHEMA CORPOREL.

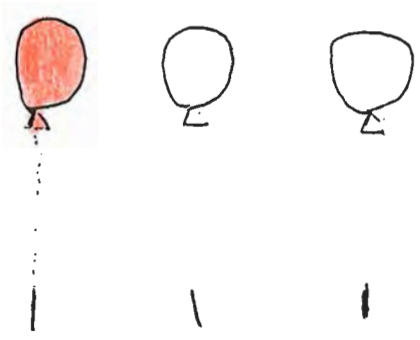
Progression : 1) Viser une partie du corps au moyen de la souris. Une fois l'élément atteint, ce même élément sur le modèle se met en couleur.
2) Reconstitution du bonhomme avec modèle.
3) Reconstitution du bonhomme sans modèle.



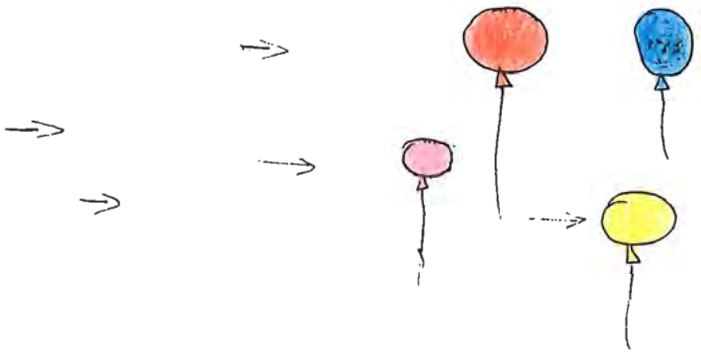


E. COORDINATION OCULO-MOTRICE.

- JEU DES FICELLES : l'élève doit continuer la ficelle. Une fois l'objectif atteint, le ballon se colorie



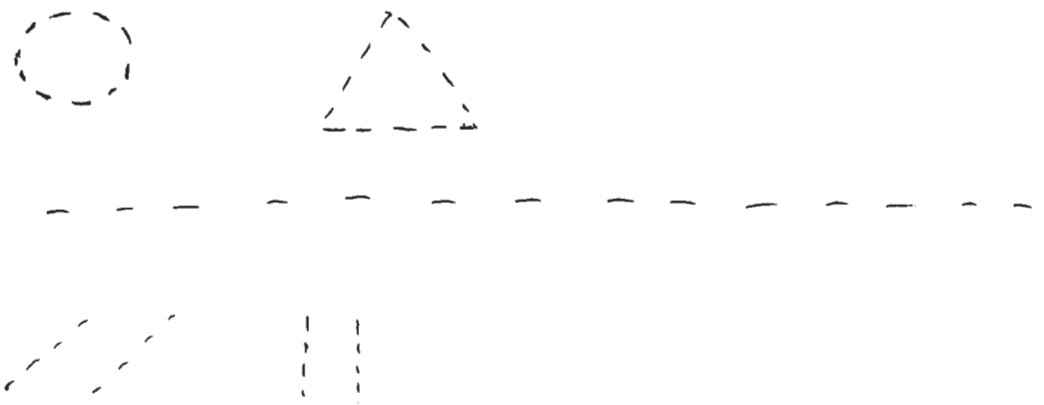
- JEU DE FLECHES : lorsque la flèche atteint le ballon, celui-ci éclate.



F. GRAPHISME.

- Achever des dessins.

- Repasser sur des formes en pointillé.



ANNEXE 2

INTRODUCTION.

Depuis plusieurs années déjà, l'Institut d'informatique des facultés Notre Dame de la Paix s'est senti concerné par le problème de l'intégration des personnes handicapées, grâce aux moyens informatiques, dans notre société.

Madame Noirhomme, très intéressée par le formidable potentiel que présente l'informatique a accepté de nous attribuer un sujet de mémoire dans ce domaine. Le premier intitulé en est "Conception de boîte à outils de programmes pour l'aide aux I.M.C.". Une description succinte du stage relatif à ce mémoire vous est proposée ci-dessous.

1° PLANNING.

Notre stage en Angleterre fut planifié de la manière suivante:

- le 2 septembre, rencontre d'information avec le Professeur Paul Golder de l'université de Birmingham.

- du 3 au 8 septembre, visites au centre de recherche H.U.S.A.T. de l'université de Laughborough.

- du 9 au 15 septembre, journées d'information au centre d'aide à la communication de l'université d'Oxford (A.C.E. center).

- du 16 au 24 septembre, visites à l'université de Birmingham.

2° PROGRAMME.

2.1° H.U.S.A.T.

A° PRESENTATION.

L'institut s'occupant essentiellement de l'aide technologique aux handicapés physiques, nous n'avons pu récolter beaucoup d'informations pertinentes. De plus, de par leurs activités, les personnes contactées, madame Colette Nicholle et madame Kate Howey, n'ont eu que peu de temps à nous consacrer.

Notre séjour à Laughborough nous a cependant permis de visiter un local d'expérimentation de logiciels et d'assister à une présentation de hardware et de software par un représentant de la firme ACORN, monsieur Vorn Hancock. Cette firme est notamment chargée de la construction et de la commercialisation des ordinateurs B.B.C.(de par leur faible coût et la multitude de périphériques disponibles sur ceux-ci, les ordinateurs B.B.C. sont les plus couramment utilisés dans les institutions et centres spécialisés dans le domaine du handicap). La présentation nous a permis de visualiser différents logiciels graphiques qui pourraient être employés par des personnes handicapées et ce, à des fins pédagogiques (notons cependant que ces logiciels ont été initialement conçus pour des enfants dits normaux).

Ajoutons encore, pour clôturer cette partie, que l'H.U.S.A.T. essaye d'avoir une influence politique sur les concepteurs et designers afin qu'ils prennent en compte le facteur "handicap physique", lors de l'élaboration de leurs projets.

B° LE MATERIEL UTILISE.

L'H.U.S.A.T. s'étant principalement concentré sur les problèmes liés à l'ergonomie du matériel pour handicapés physiques, le personnel du centre n'est pas impliqué dans l'élaboration de projets informatiques. Conséquemment, leur matériel est consitué d'un seul B.B.C. et de quelques macintosh, employés principalement à des fins administratives.

C° LES DOCUMENTS RECUS.

- "New technology work aids for the physically disables" Kate Howey 1986.

- "Factors affecting the use of information technology and computer systems as work aids for the physically disabled" Kate HOWEY 1988.

- Documents sur les logiciels présentés par Vorn Hancock pour Acorn.

D° LES ADRESSES.

- Kate Howey ou Colette Nicholle

The Elms

Elms grove

Laughborough

Leicestershire LE11 1R6

England.

Tel : 0509 611088.

- Vorn Hancock (support co-ordinator, special needs)

Acorn computers limited

Fulbourn road

Cherry Hinton

Cambridge CB1 4JN

England

Tel : 0223 245 200

2.2° A.C.E. CENTER.

A° PRESENTATION.

Ce centre, situé aux confins de la ville d'Oxford, regroupe un nombre important d'informations nécessaires aux personnes encadrant le handicapé et au handicapé en vue de l'épanouissement de celui-ci au niveau de la communication avec autrui. Il est, d'après certaines personnes, au "top niveau" dans le domaine de la technologie pour handicapés.

Il s'occupe, d'une part, de déterminer, pour un handicapé particulier, ses besoins tant au niveau matériel (hardware, système d'accès à l'ordinateur) qu'au niveau logiciel (recherche des logiciels appropriés à la déficience du handicapé). Il s'est d'autre part, grâce à d'importants subsides gouvernementaux, pleinement investi dans l'élaboration de gros projets informatiques (environ deux projets par an). Il permet également, dans le cadre de ses activités quotidiennes, la participation de personnes externes à des "training days"; journées au cours desquelles sont présentés les objectifs, le matériel et le personnel de ce centre.

C'est au cours de ces "training days" que nous avons pu participer à deux sessions d'observations de personnes handicapées évoluant dans un environnement informatique matériel et logiciel.

Notre séjour à Oxford nous a permis de rencontrer un technicien, des thérapeutes et des éducateurs avec lesquels nous avons discuté d'idées concrètes quant à la manière d'aborder la réalisation d'une application pour personnes handicapées. Nous avons pu dégager de ces conversation des points importants à prendre en considération lors de l'analyse des fonctionalités de notre application. De plus, toute une série de logiciels nous ont été présentés:

- PLOCKA et BLOB sur Archimèdes

- SAW et DRAW+ sur PC-AT (SAW sera terminé en décembre 91)

- logiciels divers réalisés par le centre

Nous avons, en outre, vu divers systèmes d'accès à l'ordinateur:

- écrans tactiles

- switches

- concept keyboards

- etc.

Notons encore que l'ACE Center a précédemment établi des contacts avec des universités étrangères. Le projet PLOCKA a notamment été réalisé avec la collaboration d'une université finlandaise. L'ACE Center a aussi laissé la réalisation de certains projets à des étudiants universitaires mais,étant donné l'insuccès de ce type de projets, le centre a décidé de laisser tomber ce genre d'expérience.

B° MATERIEL UTILISE.

Le centre n'a pas énormément de matériel. Ce qui fait la richesse de son matériel, c'est qu'il est fort diversifié. On y trouve des MAC, des BBC, des Archimèdes, des Nimbus et des PC-AT. Il dispose de plus d'un ensemble exhaustif de périphériques permettant aux divers types de personnes handicapées d'accéder à l'ordinateur.

C° DOCUMENTS RECUS.

- "The common terminology for switch controlled software" David Colven.

- "Interface mountings".

- Documents concernant les synthétiseurs de voix.

D ° ADRESSES COMMUNIQUEES.

- David Colven ACE Center

Ormerod school, Waynflete Road

Headington Oxford OX3 8DD

Tel: 0865 63508.

- ACCESS Center Hereward College of F.E.,

Bramston Crescent, Tile Hill Lane,

Coventy CV4 9SW

tel: 0203 461231 x244.

- BELFAST COMMUNICATION ADVICE Centre,

Musgrove Park Hospital,

Belfast BT9 7JB

Tel: 0232 669501.

- BRISTOL COMMUNICATION AIDS Centre,

Speech Therapy Department,

Frenchay Hospital,

Frenchay,

Bristol BS16 1LE

Tel: 0272 701212 x2151.

- CALL Centre,

4 Buccleuch Place,

Edinburgh EH8 9JT

Tel: 031 667 1438.

- CARDIFF COMMUNICATION AIDS Centre,

Rookwood Hospital, Llandaff, Cardiff,

South Glamorgan CF5 2YN

Tel: 0222 566281.

- CENMACH, Charlton Park School,

Charlton Park Road,

London SE7 8HY

Tel: 081 316 7589.

- CHARING CROSS COMMUNICATION AIDS Centre, Speech Thérapy Department, Charing Cross Hospital,

Fulham Palace Road,

London W6 8RF

Tel: 081 748 2040.

- NEWCASTLE COMMUNICATION AIDS Centre, Dene Centre,

Castle Farm Road,

Newcastle-upon-Tyne NE3 1PH

Tel: 091 284 0480.

- NORTHERN ACE, Park Dean School,

St Martin's Road,

Fitton Hill,

Oldham OL8 2PY

Tel: 061 627 1358.

- SANDWELL COMMUNICATION AIDS CENTRE est devenu CENTRE FOR HUMAN COMMUNICATION, Regional Rehabilitation Centre,

Oak Tree Lane,

Selly Oak,

Birmingham B29 6JA

Tel: 021 414 1661.

- SCTCI, Scottish Centre of Technology for the Communication Impaired, Victoria Infirmary,

Langside,

Glasgow G42 9TY

Tel: 041 649 4545 x5579/80.

- SPASTICS SOCIETY MSS,

16 Fitzroy Square,

London W1P 5HQ

Tel: 071 387 9571.

- WOLFSON COMMUNICATION AIDS CENTRE, Institute of Child Health,

The Wolfson Centre,

Mecklenburgh Square,

London WC1N 2AP

Tel: 071 837 7618.

2.3° BIRMINGHAM.

Notre séjour à Birmingham fut légèrement différent des deux précédents. Un premier contact avec le professeur Golder nous a permis de lui exprimer nos désidérata en

matière de logiciels à examiner et de contacts à prendre. Lorsque nous sommes revenus par la suite à Birmingham, ce dernier avait déjà établi un planning à notre intention.

Vous trouverez dans les paragraphes suivants une description succinte du département informatique de l'université de Birmingham et de ses activités ainsi qu'un compte-rendu détaillé de nos divers rendez-vous.

A° LE DEPARTEMENT INFORMATIQUE.

Le département informatique est localisé dans le building du département des sciences de l'université d'Aston. Plus précisément, il occupe trois ailes des trois premiers étages. A chaque étage, se trouvent divers bureaux occupés principalement par des professeurs, des assistants et des chercheurs mais certains de ces locaux peuvent être utilisés sur demande par des étudiants. En plus de ces bureaux, on y trouve des salles de classes et des pools d'ordinateurs. Le parc informatique du département est assez impressionnant: pools de MAC, de MAC II, de PC's, de terminaux Vax, de stations SUN,...

La politique du département vis-à-vis des étudiants étrangers est assez ouverte. Nous avons rencontré des étudiants de diverses nationalités réalisant un "Master In Computer Science". Ces étudiants passaient une année académique complète à Aston avec, à la clef, une reconnaissance de cette année académique de la part de leurs universités respectives.D'autres étudiants ne restaient que pour une période limitée (de 4 à 6 mois) et réalisaient des stages de même type que ceux qui nous sont proposés par l'Institut. Chose importante, tous ces stages étaient financés par des bourses Erasmus. Notons encore, pour clôturer ce paragraphe, que toute acceptation de stage en provenance de l'étranger doit se faire par l'intermédiaire du professeur Paul GOLDER, chargé des relations internationales du département informatique.

B° SNOWHILL CENTER.

La Snowhill house fait partie intégrante de l'hôpital de Birmingham. Notre visite à ce centre fut de courte durée car le personnel était en plein déménagement. Nous avons rencontré Monsieur Philippe TOWNSON . Cette personne était chargée de l'amélioration de logiciels existants de manière à les rendre plus agréables à l'utilisation . Il nous a montré quelques-uns de ses outils sur Archimedes et nous a expliqué les différents avantages de telles machines par rapport aux PC's. Malheureusement, vu l'activité débordante de ce centre, nous n'avons pu converser longtemps avec notre interlocuteur. De plus, nos centres d'intérêts ne convergent pas, notre discussion n'aura que peu ou pas de répercussion sur nos travaux.

C° LEARNING DIFFICULTIES EMPLOYMENT PREPARATION UNIT.

Nous sommes restés dans ce centre une matinée, durant laquelle nous avons pu converser avec la directrice et un professeur. Ce centre est chargé d'apprendre une activité manuelle à des personnes ayant une déficience mentale , en vue d'un insertion éventuelle dans le monde du travail. Pour ce faire, le centre s'est subdivisé en plusieurs locaux dans lesquels travaillent les handicapés sous la tutelle de leurs professeurs. Les différents apprentissages sont :

- le travail du bois,

- le travail du métal,

- le nettoyage,

- la cuisine,

- le jardinage.

Ce centre compte à ce jour 17O personnes en cours d'apprentissage. Toutes les réalisations des élèves sont vendues dans un petit magasin situé à proximité.

Ce centre développe également un programme d'étude permettant la familiarisation des étudiants avec l'environnement professionnel auquel ils seront confrontés. Pour ce faire, ils utilisent notamment l'outil informatique.

Lors de notre discussion avec un professeur, il nous est apparu qu'une des principales critiques adressées aux logiciels éducatifs existants, est le fait qu'initialement ils aient été conçus pour des enfants dits normaux. Ils ne sont donc pas adaptés à la population qui les utilise. En outre, nous avons pu constater qu'il existe dans les différentes institutions et centres, une réelle demande de logiciels permettant de guider des personnes ayant une déficience mentale dans l'environnement de tous les jours(ex. utilisation d'un mister cash,...).

D°COLLINWOOD ADULT TRAINING CENTRE.

Nous avons rencontré deux professeurs dans cette communauté:

- la première enseignante,qui s'occupe surtout des débiles profonds, nous a fait une démonstration des logiciels avec lesquels elle travaille sur un ordinateur BBC. L'entrevue

nous a permis de déterminer les points importants à prendre en considération lors de la conception d'une application informatique pour personnes ayant une déficience mentale;

- l'autre professeur s'occupait d'handicapés mentaux légers. La discussion qui s'en suivi ne fut pas très intéressante pour nos travaux, étant donné son ignorance du potentiel informatique pour l'éducation de ce type de personnes. En effet, il voyait plutôt les ordinateurs comme support de jeux pour ses "étudiants". A titre d'exemple, un handicapé mongol était chargé de recopier un livre à l'aide du traitement de texte. Cette personne ne savait ni lire, ni écrire mais cela lui permettait d'exercer une activité durant sa journée.

E° MADAME RUTH WAPPLES.

Suite à un accident, Mme Ruth WAPPLES est devenue tétraplégique. Après une période difficile, elle a décidé de suivre des cours d'informatique à l'université d'Aston. Nous avons pu parler des problèmes qu'elle rencontre dans la vie courante et des activités qu'elle exerce. Elle est devenue une militante très active dans tout ce qui touche au domaine du handicap tant du côté éducatif que du côté des problèmes de la vie courante. Elle nous a également montré le matériel sur lequel elle travaille. Il s'agit d'un Macintosh SE qu'elle manipule à l'aide d'un casque ultrasonique. Elle travaille actuellement sur un projet qui permettrait d'utiliser les symboles BLIZZ pour communiquer avec un ordinateur et/ou d'autres personnes. Mme WAPPLES a également accès à une base de donnée contenant tous les logiciels et matériels d'une marque particulière pour handicapés. Nous avons aussi pu obtenir toute une série de références de logiciels et de périphériques tournant sur des PC's pour personnes ayant une déficience mentale et/ou physique.

F° ADRESSES RECUES.

- Aston Triangle

  Paul GOLDER

  Bâtiment principal - 2éme étage - Computer Science Department

  BIRMINGHAM B4 7ET

- M.Philip Townson, Snowhill house,

  Livery street.

Cette adresse est indiquée uniquement par soucis de complétude, le centre étant en plein déménagement lors de notre visite.

- Mme Sue Hawkins, Learning Difficulties Employment Preparation Unit,

  6, Inkerma Street,

  B7 4SB Dudderston.

- Mme Fairbotham, Collinwood Adult Training Centre,

  25, Westheath Road,

  Northfield

  Tel: 477 8022.

- Mme Ruth Wapples,

  Flat 1,

  23 Milford Road,

  Harbourne.

ANNEXE 3

1.Analyse des actions.

1.1.Introduction.

Nous allons à présent spécifier les actions pouvant être associées à des objets interactifs. Les actions analysées seront celles permettant d'effectuer les traitements nécessaires à la conception d'exercices destinés à des personnes handicapées. Ces diverses actions seront donc offertes à l'informaticien comme outils d'aide à la conception et seront entièrement transparentes à l'utilisateur final, la personne ayant une déficience mentale et/ou physique.

L'analyse des logiciels précédents nous a permis de scinder quatre types d'actions:

-Le changement d'écran.
-L'apparition d'un objet.
-La disparition d'un objet.
-Le renforcement sonore.

Avant de décrire les caractéristiques spécifiques à chaque action, reprenons un peu les caractéristiques communes de ces dernières.

1.2.Description des caractéristiques communes des actions.

Conditions de déclenchement: Toute action est déclenchée soit par la sélection d'un ou plusieurs objets interactifs, soit par une action précédemment tirée. Il convient de préciser que si une liste d'actions doit être effectuée, la cause du déclenchement de la première action de cette liste ne saurait être que la sélection d'un ou plusieurs objets interactifs.

Effet: L'effet induit par la ou les condition(s) de déclenchement est, de façon évidente, le déclenchement de l'action.

Attributs abstraits communs:

-At-act-condition:attribut spécifiant les conditions devant être vérifiées afin que l'action soit exécutée.

-At-act-type:attribut spécifiant le type de l'action qui sera déclenchée. Les divers types d'actions seront (changement-écran,disparition-objet, appariton-objet, renforcement-sonore).

-At-act-identifiant:attribut permettant d'identifier l'action.

-**At-act-suivant:**attribut spécifiant si l'action à tirer est terminale ou pas.

Primitives abstraites communes:

-**Pr-act-condition:**primitive chargée de vérifier que les conditions de déclenchement de l'action sont vérifiées.

-**Pr-act-suivant:**primitive chargée de déterminer l'opportunité de la recherche d'une action suivante.

-**Pr-act-exécution:**primitive chargée d'exécuter l'action désignée.

1.3.Description des caractéristiques spécifiques à chaque action.

1.3.1.Le changement d'écran.

Attributs abstraits:

-**At-ecr-identifiant-courant:**attribut permettant d'identifier l'écran courant.

-**At-ecr-identifiant-suivant:**attribut permettant d'identifier l'écran suivant.

Primitives abstraites:

-**Pr-ecr-effacer:**primitive chargée d'effacer l'écran courant.

-**Pr-ecr-afficher:**primitive chargée d'afficher l'écran suivant.

1.3.2.L'apparition d'un objet.

Attributs abstraits:

-**At-app-objet-identifiant:**attribut permettant d'identifier l'objet devant apparaître.

-**At-app-localisation:**attribut permettant de localiser l'endroit où sera affiché l'objet.

Primitives abstraites:

-**Pr-app-objet:**primitive permettant d'afficher l'objet.

1.3.3.La disparition d'un objet.

Attributs abstraits:

-At-dis-objet: attribut permettant d'identifier l'objet qui doit disparaître.

Primitives abstraites:

-Pr-dis-objet:primitive permettant de faire disparaître un objet à l'écran.

1.3.4.Le renforcement sonore.

Attributs abstraits:

-At-son-identifiant:attribut permettant d'identifier le type de son qui sera joué.

Primitives abstraites:

-Pr-son-déclenchement:primitive chargée de faire jouer le son.

2.Analyse des opérations.

2.1.Introduction.

Outre la possibilité de définir des objets interactifs et les actions de manipulation qui y sont associées, nous avons remarqué que les logiciels précédemment élaborés contenaient tous une certaine quantité de traitements. La majorité de ces traitements opéraient sur des variables sous-jacentes aux objets affichés à l'écran. Nous avons décidé de rendre possible un mécanisme de définition de variables ainsi que toute une série d'opérations sur ces dernières. A tout objet de type icône sera associé un attribut permettant de spécifier si une variable y est attachée et si oui, quel est son type et sa valeur s'il y a lieu.

La définition d'une opération sera similaire à celle d'une action de telle sorte que l'on pourra inclure une opération dans une suite d'actions au même titre que ces dernières.

Notons pour terminer que toute opération contiendra une primitive qui sera celle effectuant le traitement pour lequel elle est définie. Mais, dans le cas des tests, nous aurons une primitive supplémentaire qui sera chargée de déclencher l'action

dépendante du résultat du test. Si une des deux actions n'est pas définie cela signifie que, dans ce cas, il ne faut rien faire.

2.2.L'opération: test d'égalité.

Définition:cette opération permet de tester si deux expressions sont égales. Si oui, la première action définie sera tirée. Sinon, ce sera la seconde.

Attributs abstraits:

-At-éga-nom-var1:attribut spécifiant le nom de la première variable.

-At-éga-nom-var2:attribut spécifiant le nom de la deuxième variable s'il y a lieu. En effet, toute comparaison peut s'effectuer entre le contenu d'une variable et une valeur. Nous n'aurons donc pas nécessairement deux noms de variables dans les attributs de cette opération.

-At-éga-type-variable: attribut spécifiant de quel type seront la ou les expressions sur lesquelles portera l'opération. Les différents types seront = (entier, string, virgule_flottante).

-At-éga-identifiant-action1: attribut permettant,s'il y a lieu, d'identifier l'action qui sera tirée en cas de vérification du test.

-At-éga-identifiant-action2: attribut permettant,s'il y a lieu, d'identifier l'action qui sera tirée en cas de non-vérification du test.

2.3.L'opération: test du plus petit.

Définition: cette opération permettra de tester si entre deux expressions l'une est plus petite(ou plus petite ou égale) que l'autre. Si oui, la première action sera tirée sinon, la seconde.

Attributs abstraits:

-At-ppt-nom-var1:attribut spécifiant le nom de la première variable.

-At-ppt-nom-var2:attribut spécifiant le nom de la deuxième variable s'il y a lieu. En effet, toute comparaison peut s'effectuer entre le contenu d'une variable et une valeur. Nous n'aurons donc pas nécessairement deux nom de variables dans les attributs de cette opération.

-At-ppt-type-variable: attribut spécifiant de quel type seront la ou les expressions sur lesquelles portera l'opération. Les différents types seront = (entier, string, virgule_flottante).

-At-ppt-identifiant-action1: attribut permettant,s'il y a lieu, d'identifier l'action qui sera tirée en cas de vérification du test.

-At-ppt-identifiant-action2: attribut permettant,s'il y a lieu, d'identifier l'action qui sera tirée en cas de non-vérification du test.

-At-ppt-égal: attribut booléen spécifiant s'il s'agit du test plus petit ou égal.

2.4.L'opération: test du plus grand.

Définition: cette opération permet de tester si entre deux expresions une est plus grande(ou plus grande ou égale) que l'autre. Si oui, la première action définie sera tirée sinon ce sera la seconde.

Attributs abstraits:

-At-pgd-nom-var1:attribut spécifiant le nom de la première variable.

-At-pgd-nom-var2:attribut spécifiant le nom de la deuxième variable s'il y a lieu. En effet, toute comparaison peut s'effectuer entre le contenu d'une variable et une valeur. Nous n'aurons donc pas nécessairement deux nom de variables dans les attributs de cette opération.

-At-pgd-type-variable: attribut spécifiant de quel type seront la ou les expressions sur lesquelles portera l'opération. Les différents types seront = (entier, string, virgule_flottante).

-At-pgd-identifiant-action1: attribut permettant,s'il y a lieu, d'identifier l'action qui sera tirée en cas de vérification du test.

-At-pgd-identifiant-action2: attribut permettant,s'il y a lieu, d'identifier l'action qui sera tirée en cas de non-vérification du test.

-At-pgd-égal: attribut spécifiant s'il s'agit du test plus grand ou égal.

2.5.L'opération: soustraction de deux expressions.

Définition: cette opération permet de soustraire d'une expression la valeur d'une seconde.

Attributs abstraits:

-At-sou-nom-var1:attribut spécifiant le nom de la première variable.

-At-sou-nom-var2:attribut spécifiant le nom de la deuxième variable s'il y a lieu. En effet,cette opération peut s'effectuer entre le contenu d'une variable et une valeur.

-At-sou-type-variable: attribut spécifiant de quel type seront les expressions sur lesquelles portera l'opération. Les différents types seront = (entier, virgule_flottante).

-At-sou-nom-var-destination:attribut spécifiant le nom de la variable qui contiendra le résultat de l'opération.

2.6.L'opération: addition de deux expressions.

Définition: cette opération permet d'additionner deux expressions.

Attributs abstraits:

-At-add-nom-var1:attribut spécifiant le nom de la première variable.

-At-add-nom-var2:attribut spécifiant le nom de la deuxième variable s'il y a lieu. En effet,cette opération peut s'effectuer entre le contenu d'une variable et une valeur.

-At-add-type-variable: attribut spécifiant de quel type seront les expressions sur lesquelles portera l'opération. Les différents types seront = (entier, virgule_flottante).

-At-add-nom-var-destination:attribut spécifiant le nom de la variable qui contiendra le résultat de l'opération.

2.7.L'opération: multiplication de deux expressions.

Définition: cette opération permet de multiplier deux expressions.

Attributs abstraits:

-At-mul-nom-var1:attribut spécifiant le nom de la première variable.

-At-mul-nom-var2:attribut spécifiant le nom de la deuxième variable s'il y a lieu. En effet,cette opération peut s'effectuer entre le contenu d'une variable et une valeur.

-At-mul-type-variable: attribut spécifiant de quel type seront les expressions sur lesquelles portera l'opération. Les différents types seront = (entier, virgule_flottante).

-At-mul-nom-var-destination:attribut spécifiant le nom de la variable qui contiendra le résultat de l'opération.

2.8.L'opération: division de deux expressions.

Définition: cette opération permet la division d'une expression par une autre.

Attributs abstraits:

-At-div-nom-var1:attribut spécifiant le nom de la première variable.

-At-div-nom-var2:attribut spécifiant le nom de la deuxième variable s'il y a lieu. En effet,cette opération peut s'effectuer entre le contenu d'une variable et une valeur.

-At-div-type-variable: attribut spécifiant de quel type seront les variables sur lesquelles portera l'opération. Les différents types seront = (entier, virgule_flottante).

-At-div-nom-var-destination:attribut spécifiant le nom de la variable qui contiendra le résultat de l'opération.

2.9.L'opération: concaténation de deux expressions.

Définition: cette opération permet de concaténer deux strings.

Attributs abstraits:

-At-cct-nom-var1:attribut spécifiant le nom de la première variable.

-At-cct-nom-var2:attribut spécifiant le nom de la deuxième variable s'il y a lieu. En effet,cette opération peut s'effectuer entre le contenu d'une variable et une valeur.

-At-cct-nom-var-destination:attribut spécifiant le nom de la variable qui contiendra le résultat de l'opération.

2.10.L'opération: assignation d'une valeur à une variable.

Définition: cette opération permet d'attribuer une valeur à une variable.

Attributs abstraits:

-At-ass-nom-var1:attribut spécifiant le nom de la variable s'il y a lieu. En effet,cette opération peut s'effectuer avec le contenu d'une variable ou une valeur.

-At-ass-nom-var-destination:attribut spécifiant le nom de la variable qui contiendra le résultat de l'opération.

-At-ass-type-variable: attribut spécifiant de quel type seront les variables sur lesquelles portera l'opération. Les différents types seront = (entier, string, virgule_flottante).