MASTER EN SCIENCES INFORMATIQUES

Conception et réalisation d'un Éditeur Graphique Orienté Objet

Collin, Jean-Marie

Award date:
1992

Awarding institution:
Universite de Namur

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

Conception et réalisation d'un Editeur Graphique Orienté Objet

par Jean-Marie COLLIN

Promoteur :

François Bodart

Mémoire présenté en vue de
l'obtention du titre de
Licencié et Maître en Informatique

RUE GRANDGAGNAGE, 21, B - 5000 NAMUR (BELGIUM)

Facultés Universitaires Notre-Dame de la Paix

Institut d'Informatique

Rue Grandgagnage, 21 B

B-5000 NAMUR

Conception et réalisation d'un Editeur Graphique Orienté Objet

Jean-Marie COLLIN

Résumé :

Les modèles utilisés pour la conception de systèmes d'informations et pour le développement de logiciels se représentent assez généralement sous forme de graphes. Il s'avère alors particulièrement intéressant de disposer d'un outil logiciel capable de construire de telles représentations graphiques de modèles, de manière interactive, en manipulant simplement les éléments qui les composent. Il est également nécessaire que l'outil soit assez général pour être indépendant du type de modèle qu'il doit représenter. L'éditeur graphique orienté objet EGOO, conçu et implanté dans le cadre de ce mémoire, répond à ces objectifs. Développé en C++ sous MS-Windows, EGOO espère satisfaire les besoins de tous les concepteurs et développeurs en matière d'éditeur graphique.

Abstract :

Models used for information systems conception and for software development generaly have a graphical representation. So, it's highly interesting to have a software for the interactive creation of such graphic representations by using graphic components. It's also necessary to have a generic software independent of the model to be represented. The "Editeur Graphique Orienté Objet EGOO" (created here) gives you the solution. Made with C++ under MS-Windows, EGOO is a response to what creators and developpers needed.

Mémoire présenté en vue de l'obtention du titre

de Licencié et Maître en Informatique

Septembre 1992

Promoteur : François Bodart

Je tiens à remercier

Monsieur le Professeur François Bodart, qui m'a permis de réaliser ce mémoire,

Monsieur Benoît Sacré, qui m'a suivi tout au long de ce travail et m'a prodigué de judicieux conseils,

Mademoiselle Françoise Warnier, qui m'a supporté lors de la réalisation de ce mémoire et m'a fourni une aide précieuse lors de la conception du logiciel et de la rédaction du mémoire,

Monsieur Paul Bawin, avec qui de passionnantes discussions m'ont permis d'approfondir mes connaissances et engendrer des réflexions sur l'Orienté Objet et sur MS-Windows,

à mes parents, sans qui je n'aurais pu réaliser ces études.

Table des matières

Chapitre 1

1. Introduction

De nombreux modèles de conception des systèmes d'informations et de développement de logiciels peuvent être représentés sous forme graphique. Une perspective intéressante pour la réalisation de ces représentations graphiques de modèles est de pouvoir les construire en manipulant, de manière interactive, les éléments graphiques qui les composent. Il serait donc utile d'avoir un outil permettant la manipulation directe d'éléments graphiques afin de construire aisément les schémas et de pouvoir les modifier sans trop de difficultés.

Le premier objectif de ce travail est de réaliser le logiciel EGOO (Editeur Graphique Orienté Objet). Ce logiciel est un éditeur graphique, développé sous l'environnement Microsoft Windows (noté MS-Windows), permettant la construction de représentations graphiques de modèles de conception des systèmes d'informations et de développement de logiciels.

Un deuxième objectif concerne l'évaluation de l'approche orientée objet, utilisée pour le développement du logiciel EGOO sous l'environement MS-Windows.

Le logiciel EGOO est basé sur une architecture client-serveur. L'éditeur graphique (client) ne maintient que l'information graphique des schémas de modèles qu'il représente et communique avec un serveur pour obtenir certaines informations. Le serveur, pour sa part, maintient toute la sémantique des schémas.

L'implantation du logiciel EGOO est réalisée en C++. Au moment de la réalisation de cette implantation, il a fallu faire un choix relatif aux interactions entre le logiciel EGOO et MS-Windows. En effet, à ce moment, l'appel des fonctions du SDK (Software Development Kit) de MS-Windows ne pouvait être réalisé qu'en C. La version C++ du SDK n'existait pas.

Nous avions deux possibilités pour l'appel des fonctions de l'environnement MS-Windows :

- Utiliser une hiérarchie de classes décrivant l'environnement MS-Windows comme une collection d'objets ,

- Appeler les fonctions du SDK en C classique.

La première possibilité permet de travailler avec un MS-Windows "orienté objet". Au moment de la réalisation du logiciel, les classes proposées par différentes firmes décrivant MS-Windows n'étaient qu'une "sur-couche" du SDK. Autrement dit, les classes étaient implantées en utilisant les fonctions C classiques du SDK et ne faisaient pas directement appel

à l'environnement MS-Windows. De plus, toutes les firmes proposant des classes de MS-Windows, avaient leur propre hiérarchie de classes. Aucun standard n'était défini.

En raison de la " sur-couche " du SDK et de la non standardisation des classes, nous avons opté pour la deuxième possibilité de l'interaction entre le logiciel EGOO et MS-Windows, à savoir l'appel des fonctions du SDK en C classique.

Le chapitre 2 décrit les diverses fonctionnalités qui sont demandées au logiciel EGOO. Ce chapitre exprime les besoins de l'éditeur graphique.

La conception du logiciel est réalisée dans le chapitre 3. Nous y décrirons la modélisation abstraite des représentations graphiques de schémas que le logiciel doit être capable de représenter. Nous développerons également la communication entre EGOO et son serveur, ainsi que la conception orientée objet du logiciel.

Le chapitre 4 sera consacré à l'implantation du logiciel. Nous y décrirons l'implantation de toute l'architecture de l'éditeur graphique ainsi que les classes d'objets constituant cette architecture.

Enfin, nous terminerons par le chapitre 5 qui apportera une conclusion du travail réalisé.

Chapitre 2

2. Fonctionnalités du logiciel

Le logiciel EGOO est un éditeur graphique de manipulation de " graphes " . Les graphes manipulés, en deux dimensions, sont constitués de noeuds et d'arcs. Les arcs portent sur des noeuds ou sur d'autres arcs.

Cet éditeur graphique doit permettre de manipuler les représentations graphiques de modèles utilisés pour la conception des systèmes d'informations (modèle de la méthode IDA, Modèle d'Accès Généralisé, ...) ainsi que des modèles utilisés pour le développement de logiciels (organigrammes, diagrammes de Nassi-Schneiderman, ...).

L'éditeur graphique doit être général pour permettre de manipuler les représentations graphiques de plusieurs modèles sans avoir à modifier son code pour l'adapter à chaque nouveau modèle. Il doit donc être conçu avec une grande flexibilité sur le type de représentation graphique qu'il peut manipuler.

Dans la première partie de ce chapitre, nous allons définir un éditeur graphique en général et donner les différentes fonctionnalités qu'il doit offrir.

Pour pouvoir étudier les différents types de représentations graphiques que le logiciel EGOO doit pouvoir représenter et en dégager les comportements, nous allons nous baser sur les représentations graphiques de modèles qui nous sont familiers, à savoir les modèles de la méthode IDA et le Modèle d'Accès Généralisé. Ces modèles sont décrits dans la deuxième partie de ce chapitre.

Dans une troisième partie, nous étudierons les différentes propriétés graphiques des modèles. Cette analyse nous permettra de nous rendre compte des nombreuses difficultés que nous serons susceptibles de rencontrer lors de l'élaboration du logiciel EGOO pour la représentation de ces modèles.

Nous terminerons par une description détaillée et complète des fonctionnalités du logiciel EGOO. Autrement dit, dans la dernière partie, nous exprimerons les besoins du logiciel.

2.1. Editeur graphique

Cette partie présente les fonctionnalités générales d'un éditeur graphique.

Nous pouvons définir un éditeur graphique de la manière suivante :

Un éditeur graphique est un logiciel permettant la manipulation interactive directe d'éléments grâce à leur représentation graphique en vue de construire un schéma. Un schéma est défini comme un agrégat d'éléments de base. Ces derniers sont les "briques de base" mises à la disposition de l'utilisateur par le logiciel.

Un éditeur graphique est donc un logiciel permettant de manipuler des représentations graphiques d'éléments, tout comme nous le ferions sur une feuille de papier (mis à part le fait que l'interaction entre l'utilisateur et le logiciel pour manipuler des éléments graphiques est beaucoup plus souple qu'entre l'utilisateur et une feuille de papier).

Lorsque nous voulons "dessiner" la représentation graphique d'un modèle à l'aide d'un éditeur graphique, il suffit de prendre les représentations graphiques des éléments de base (ceux-ci pouvant être simples - par exemple : droite, courbe, polygone, ... - ou complexes - par exemple : un ensemble de droites et de textes définissant un élément du modèle à représenter, ... - en fonction du modèle à représenter) et de les disposer pour qu'ils forment le dessin voulu. A tout moment, les éléments du dessin peuvent être modifiés vu qu'ils restent des éléments en tant que tels et ne deviennent pas de simples images "bitmap".

Les exemples suivants illustrent des éditeurs graphiques :

- Dessin vectoriel : c'est un éditeur graphique qui permet de construire des "dessins artistiques". Les éléments qui peuvent être utilisés pour composer le dessin sont les suivants : texte, droite, rectangle, polygone, courbe, ... L'utilisateur voulant dessiner, prend un à un les éléments de base nécessaires, les dispose sur son dessin, et les arrange les uns par rapport aux autres. Un exemple de dessin est donné à la figure 1.

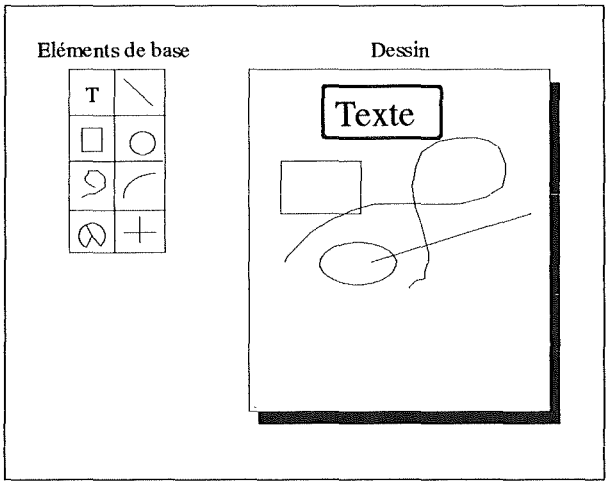

Fig. 1 : Dessin vectoriel

- Editeur de graphes : c'est un éditeur graphique qui permet de construire des graphes (les graphes sont composés de noeuds et d'arcs les reliant). L'utilisateur dispose des éléments de base - une droite, une courbe, ... pour un arc et un cercle pour un noeud - pour assembler le graphe. Un exemple de graphe est donné à la figure 2.

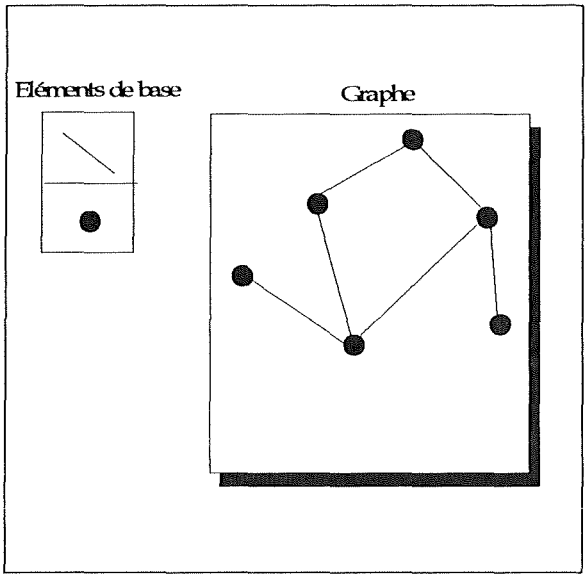

Fig. 2 : Dessin d'un éditeur de graphes

Tout éditeur graphique doit pouvoir offrir les fonctionnalités suivantes :

- Insérer dans le schéma[1] graphique des éléments de base,
- Déplacer sur le schéma graphique des éléments,
- Supprimer des éléments du schéma graphique,
- Modifier des éléments du schéma graphique,
- Redimensionner en taille des parties du schéma graphique,
- Modifier la vue du schéma graphique sur lequel l'utilisateur est en train de travailler,
- Effectuer un zoom sur une partie du schéma graphique.

2.2. Modèles à représenter

Cette partie expose les différentes représentations graphiques d'exemples de modèles sur lesquels nous nous basons pour l'élaboration du logiciel EGOO.

Nous allons, par la suite, énumérer les différents modèles et leurs représentations graphiques (pour plus de précision sur les modèles et leurs représentations graphiques le lecteur se reportera à la bibliographie) :

Modèle Entité-Association (E/A) ([1])

Le modèle E/A a une représentation graphique composée des cinq éléments de base suivants : type d'entité (T.E.), type d'association (T.A.), attribut, rôle reliant un T.E. à un T.A. et contrainte d'intégrité (C.I.) usuelle possédant une représentation graphique significative.

Une illustration de la représentation graphique de ce modèle est donnée à la figure 3.

1 Le terme *schéma* employé ici n'est pas utilisé dans son sens habituel. Il sera défini ultérieurement.

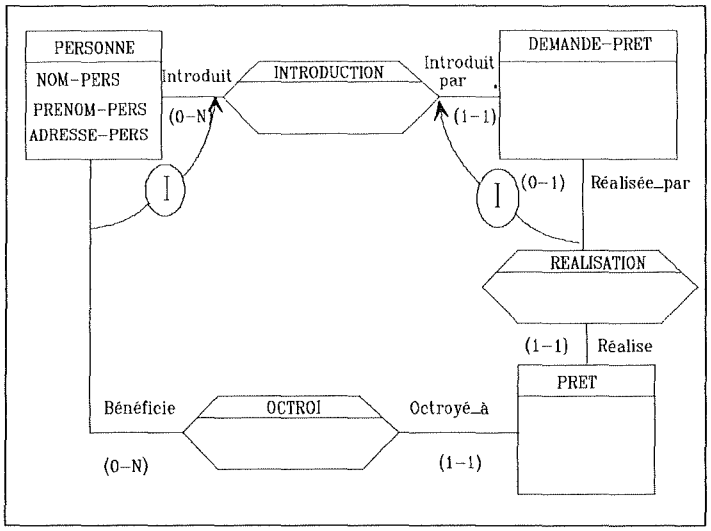

Fig. 3 : Schéma du modèle E/A

Modèle de structuration des traitements ([1])

Le modèle de structuration des traitements a une représentation graphique composée de chaînes de caractères et d'arcs les reliant.

Une illustration de la représentation graphique de ce modèle est donnée à la figure 4.

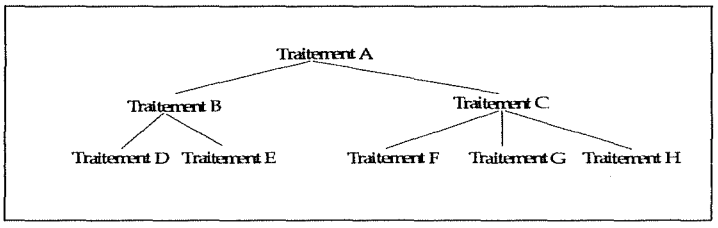

Fig. 4: Schéma du modèle de structuration des traitements

Modèle de la dynamique des traitements ([1])

Le modèle de la dynamique des traitements a une représentation graphique composée de processus, messages, points de synchronisation, conditions de sélection, mécanismes de duplication et arcs reliant les éléments précédemment cités.

Une illustration partielle de la représentation graphique de ce modèle est donnée à la figure 5.

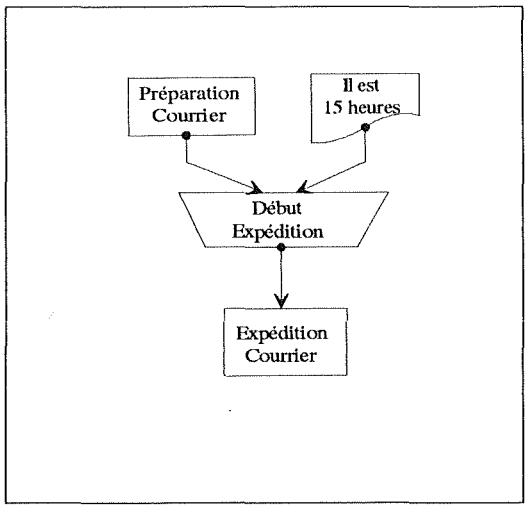

Fig. 5: Schéma de la dynamique des traitements

Modèle du diagramme des flux ([1])

Le modèle du diagramme des flux a une représentation graphique composée d'unités organisationnelles, de messages, de mémoires, de traitements, d'environnements du S.I. ainsi que d'arcs reliant les quatre derniers éléments précédemment cités.

Une illustration de la représentation graphique de ce modèle est donnée à la figure 6.

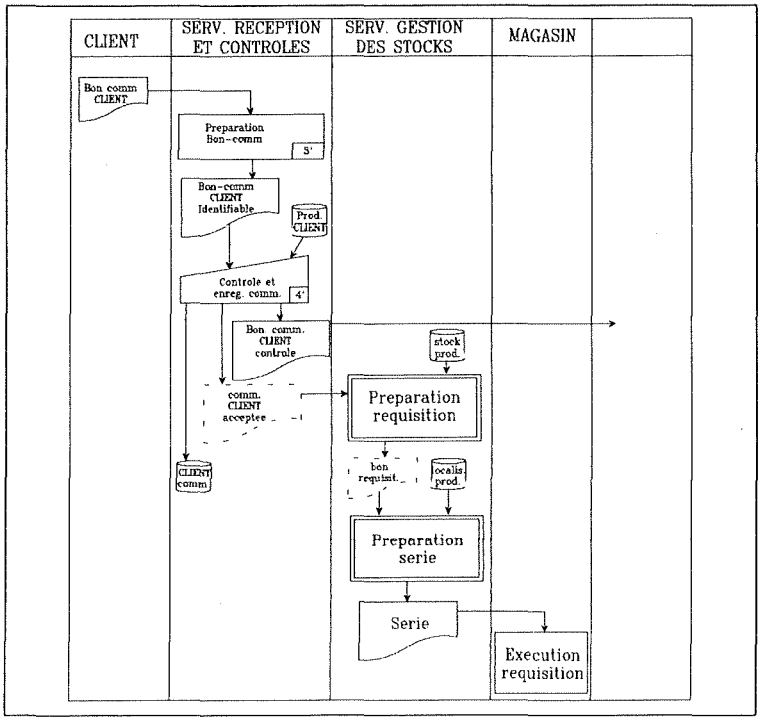

Fig. 6 : Schéma du diagramme des flux

Modèle MAG ([2])

Le modèle MAG a une représentation graphique composée de types d'articles, d'items et de types de chemins les reliant.

Une illustration partielle de la représentation graphique de ce modèle est donnée à la figure 7.

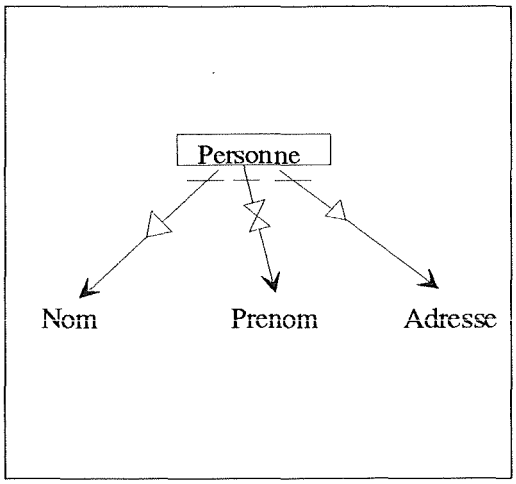

Fig. 7 : Schéma du modèle MAG

2.3. Analyse des propriétés graphiques des modèles

Cette partie illustre les principales difficultés rencontrées pour l'élaboration des représentations graphiques des modèles cités au point précédent.

Nous allons, par la suite, illustrer point par point ces différentes difficultés.

2.3.1. Arcs complexes

Dans la plupart des représentations graphiques des modèles que le logiciel EGOO doit pouvoir gérer, un arc, c'est-à-dire un lien entre deux objets du schéma, n'est pas simplement constitué d'une droite. L'arc portant sur deux objets du schéma est constitué au minimum d'une droite ou d'une courbe. Mais l'arc peut être très complexe, et sur cette droite ou cette courbe peuvent venir se disposer d'autres éléments géométriques (droite, triangle, diabolo, cercle, ...) ou des éléments de texte.

Pour illustrer cela, prenons par exemple le modèle E/A. Un arc désigne un rôle, c'est-à-dire un lien entre un T.E. et un T.A.. Il est donc composé d'une droite, mais aussi de deux chaînes de caractères (la connectivité et le nom du rôle). Il en est de même pour les C.I. ayant une

représentation graphique (par exemple : une contrainte d'inclusion est définie par une droite ou une courbe ainsi que par un rond contenant la lettre "I").

Nous pouvons aussi trouver un exemple dans le modèle MAG où un arc désigne un chemin, c'est-à-dire un lien entre deux articles ou entre un article et un item. Dans ce cas, l'élément de base de l'arc est une droite mais cet arc peut aussi comprendre un triangle, un diabolo, ...

Une illustration de ces arcs complexes est donnée aux figures 8 et 9.

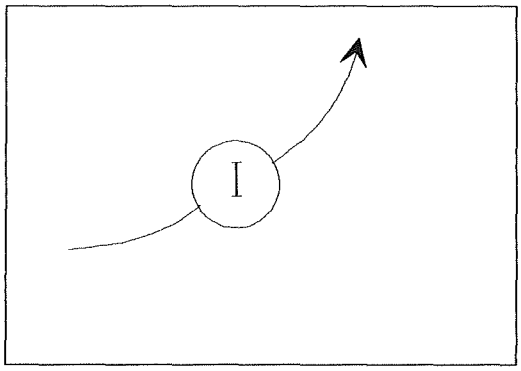

Fig. 8 : Arc complexe du modèle E/A

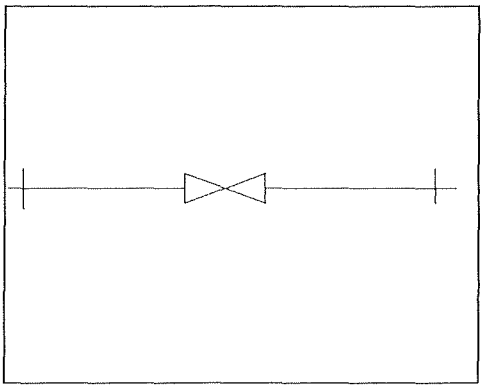

Fig. 9 : Arc complexe du modèle MAG

Les arcs peuvent donc être des éléments très complexes et l'utilisateur doit avoir la possibilité de pouvoir les déplacer, les insérer, les effacer et les redimensionner en taille. Toutes ces opérations, qu'il doit être possible d'effectuer sur un arc, doivent être réalisées sur l'arc en tant que tel. En effet, l'arc représentant un concept du modèle, toutes les opérations effectuées sur l'arc sont, en fait, des opérations effectuées sur l'élément arc et non sur les éléments constituant l'arc. Une opération réalisée sur un arc ne doit donc pas être décomposée en opérations portant sur chacun des éléments composant l'arc, mais doit être réalisée sur l'arc en tant qu'élément complexe.

De plus, un déplacement d'un des deux objets du schéma sur lequel porte un arc entraîne le déplacement de l'arc pour qu'il garde son lien avec l'objet déplacé.

L'opération de déplacement d'un objet sur lequel porte un arc peut être illustré en prenant le modèle E/A :

Un rôle portant sur un T.E. et un T.A. est un arc. Le déplacement du T.E. entraîne le déplacement de l'arc le reliant au T.A. pour que les extrémités de l'arc se situent toujours sur le T.E. et le T.A. en question.

2.3.2. Arcs définis sur deux arcs

Si nous définissons un arc comme étant un élément représentant un lien entre deux objets du schéma, nous devons pouvoir définir un arc portant sur deux autres arcs.

Prenons pour exemple le modèle E/A. Une C.I. graphique peut porter sur deux arcs reliant des T.E. et T.A.. En particulier, une contrainte d'inclusion de rôles a pour représentation graphique une flèche partant d'un arc reliant un T.E. à un T.A. vers un autre arc reliant le T.E. en question à un autre T.A. . De plus, cette flèche supporte un cercle dans lequel figure la lettre "I". Cette C.I. graphique est illustrée à la figure 3.

Alors que les arcs représentant des rôles portent sur des T.E. et des T.A., qui sont des éléments complexes, les contraintes d'inclusion de rôles, quant à elles, portent sur deux arcs, autrement dit, deux droites ou deux courbes. Lors d'un déplacement d'un arc sur lequel porte une contrainte d'inclusion de rôles, la représentation graphique de cette dernière doit aussi être déplacée en même temps que l'arc qui la supporte.

2.3.3. Rotations des arcs

Comme nous l'avons défini dans le point intitulé "Arcs complexes" (Cf. 2.3.1.) pour le modèle E/A, un arc est un élément plus ou moins complexe car il est composé d'une droite ou d'une courbe et d'autres éléments géométriques. Lors de la construction d'un schéma, un arc doit pouvoir subir une rotation. Ce principe de rotation n'est pas aussi simple que nous pourrions l'imaginer. En effet, tous les éléments de l'arc n'effectueront pas nécessairement cette rotation. Prenons par exemple un arc représentant une contrainte d'inclusion. Lors de sa rotation, tous les éléments composant l'arc doivent subir la rotation, excepté la lettre I. Cette caractéristique est représentée à la figure 10 où la figure 10.(a) représente l'arc avant rotation, la figure 10.(b) représente l'arc après rotation de tous ses composants, et enfin en 10.(c) l'arc a subi une rotation de tous ses composants sauf de la lettre "I".

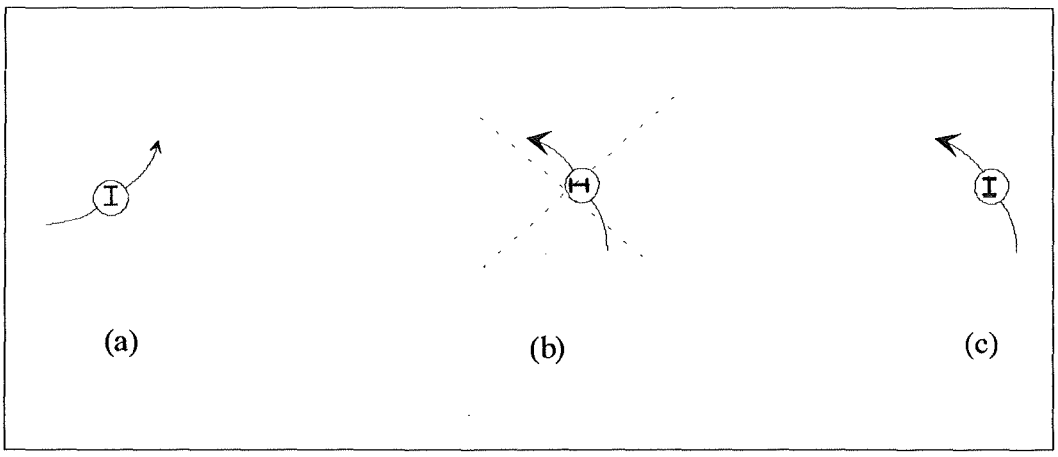

Fig. 10 : Rotation d'une contrainte d'inclusion

Par contre, si nous considérons la rotation d'un arc du modèle MAG possédant un diabolo en son milieu, tous les éléments de l'arc effectueront la rotation, y compris le diabolo. Cette caractéristique est représentée à la figure 11 où la figure 11.(a) représente l'arc avant rotation, la figure 11.(b) représente l'arc après rotation uniquement de la droite définissant l'arc, et enfin en 11.(c) l'arc a subi une rotation de tous ses composants.

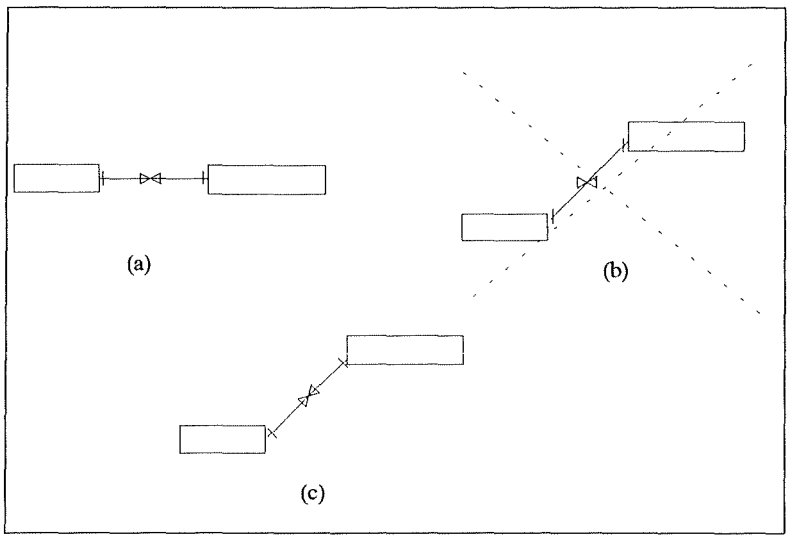

Fig. 11 : Rotation d'un arc du modèle MAG

Comme nous l'avons illustré dans les deux exemples précédents, les différents éléments composant un arc doivent donc avoir la possibilité de subir ou non la rotation appliquée à l'arc.

2.3.4. Contraintes sur des éléments d'un arc

Lors d'un redimensionnement en taille d'un arc, certains composants doivent pouvoir garder la même "position". Prenons le cas du modèle MAG et considérons un arc entre deux articles représentant un chemin obligatoire des deux "côtés". Cet arc est donc composé d'une droite et de deux petits traits verticaux placés près des deux extrémités de l'arc.

Si un redimensionnement est effectué sur cet arc dans le sens horizontal, les deux barres verticales vont être déplacées de façon à ce qu'elles conservent la proportion des distances par rapport aux extrémités de l'arc. Cette situation nous pose quelques problèmes. Illustrons à la figure 12 .(a) l'arc de départ et à la figure 12 .(b) l'arc après un redimensionnement. Lors de ce dernier, les barres représentatives du caractère obligatoire se sont déplacées et ne se trouvent plus du tout proche des deux extrémités de l'arc, ce qui est dépourvu de sens.

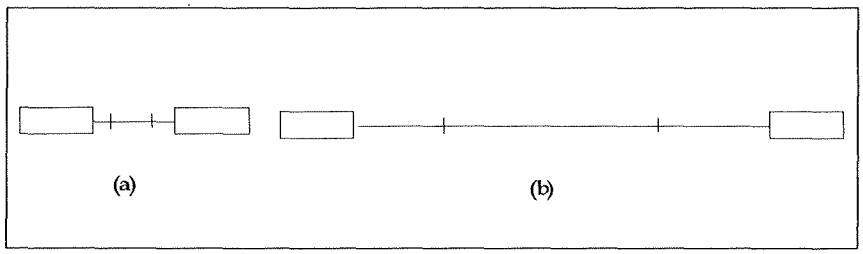

Fig. 12 : Redimensionnement normal

Il faut donc pouvoir imposer que ces deux petites barres puissent toujours exprimer la signification que nous leur donnons. Autrement dit, elles doivent toujours se trouver à une certaine distance des extrémités de l'arc. Un exemple de redimensionnement correct est illustré à la figure 13 où nous retrouvons en 13.(a) l'arc avant redimensionnement et en 13.(b) l'arc après redimensionnement correct.

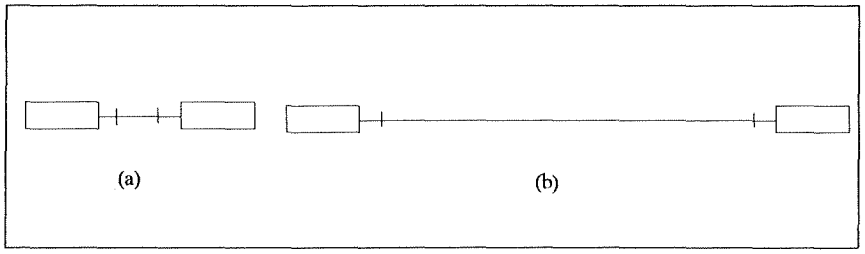

Fig. 13 : Redimensionnement correct

2.4. Fonctionnalités du logiciel EGOO

Cette partie présente les fonctionnalités demandées au logiciel EGOO. Ces fonctionnalités sont énumérées et décrites les unes après les autres.

Nous allons tout d'abord donner quelques définitions :

Modèle :
Un modèle est formé de concepts et de relations entre ces concepts.

Schéma :
Un schéma est une description d'une réalité à l'aide d'un modèle dans un langage donné.

Schéma graphique :
Un schéma graphique est un schéma dont le langage est un langage graphique.

Représentation graphique :
Une représentation graphique est un ensemble de conventions graphiques pour la représentation de concepts qui met en évidence la structuration des concepts d'un modèle.

Vue :
Une vue est une partie visible d'un schéma.

Serveur :
Un serveur est une application disposant de données qui peuvent être consultées, modifiées, ajoutées ou supprimées par d'autres applications (dénommées clients). Les données restent toujours sous le contrôle du serveur.

Elément graphique :
Tout élément visuel apparaissant sur un schéma graphique est considéré comme un élément graphique. Cet élément possède donc un comportement.

Graphe sémantique :
Un graphe sémantique est un ensemble d'informations représentant la sémantique relative aux éléments d'un schéma.

Remarque : Par abus de notation, nous désignerons schéma graphique par schéma.

2.4.1. Fonctions classiques d'un éditeur graphique

Le logiciel EGOO, étant un éditeur graphique, doit offrir toutes les fonctions classiques que fournit un éditeur graphique. Il doit donc pouvoir offrir des fonctions sur les éléments de base - insérer des éléments sur le schéma, les effacer, les déplacer, les modifier, les redimensionner en taille, ... - ainsi que des fonctions générales sur le schéma - fonctions de zoom, déplacement de la vue du schéma, ... -. Pour rappel, la description générale d'un éditeur

graphique et de ses fonctionnalités a été étudiée avec plus de détails dans le point intitulé "Editeur graphique" (Cf. 2.1.).

2.4.2. Manipulation d'éléments graphiques complexes

Le logiciel EGOO doit offrir à l'utilisateur des éléments graphiques de structure complexe pour la construction d'un schéma.

Pour construire un schéma, un ensemble d'éléments graphiques doit être mis à la disposition de l'utilisateur. Les éléments graphiques manipulés par l'utilisateur doivent être aussi proches que possible de la signification conceptuelle qu'ils représentent. Pour cela, les éléments graphiques peuvent être de structure complexe. Prenons par exemple le cas du modèle E/A. Les éléments graphiques qui doivent être offerts à l'utilisateur sont les T.E., T.A., rôles et contraintes d'intégrité. L'élément graphique représentant par exemple un T.E. est composé d'un rectangle et d'une droite dans sa partie supérieure, d'un nom désignant le nom du T.E. ainsi qu'une liste de noms représentant les attributs du T.E.. Cet élément graphique a donc une structure complexe contenant rectangle, droite et chaîne de caractères.

Ces éléments graphiques pouvant avoir une structure complexe, nous devons être capables de les manipuler en tant qu'un seul élément, mais nous devons aussi pouvoir manipuler séparément tous les éléments qui les composent (par exemple, nous devons avoir la possibilité de redimensionner le nom du T.E., ...).

2.4.3. Logiciel général

Le logiciel doit pouvoir être suffisamment général pour permettre la représentation de schémas très divers. Il doit être possible de représenter aussi bien un schéma du modèle E/A, qu'un schéma du modèle diagramme des flux, ... (des exemples de modèles que le logiciel doit être capable de représenter ont été donnés dans le point 2.2. intitulé "Modèles à représenter").

2.4.4. Utilisation d'un serveur

Le programme EGOO est un éditeur graphique pour la construction de schémas et ne maintient que l'information graphique de ceux-ci. Il doit donc offrir un dialogue avec un serveur conservant d'autres informations. Ce serveur est une application qui communique avec son client, le programme EGOO, grâce à un protocole de communication.

Lors de la construction d'un schéma, l'éditeur graphique maintient toute l'information quant à la représentation et la disposition des éléments contenus dans le schéma. Le serveur maintient pour sa part d'autres informations à propos des éléments qui sont contenus dans ce même schéma ainsi qu'à propos des relations entre ces éléments. L'information maintenue par le serveur sur les éléments d'un schéma peut être beaucoup plus riche que ce qui est représenté des éléments par l'éditeur graphique. Par exemple, pour un schéma E/A, si l'éditeur graphique

ne représente pour un T.E. que les éléments graphiques (rectangle et droite, le nom du T.E. et les noms des attributs), le serveur doit maintenir l'information sur certaines caractéristiques des attributs : simples ou répétitifs, élémentaires ou décomposables, obligatoires ou facultatifs, ...

Pour chaque schéma représenté par l'éditeur graphique, le serveur possède le graphe sémantique qui lui correspond. Autrement dit, chaque fois que l'utilisateur crée un schéma via l'éditeur graphique, un graphe sémantique est aussi créé par le serveur et correspond au schéma de l'éditeur graphique. De plus, nous pouvons dire qu'à un instant donné il y a une correspondance biunivoque entre les éléments d'un graphe sémantique ayant une représentation graphique stockés par le serveur et ceux maintenus par l'éditeur graphique pour le schéma correspondant. Le serveur est garant de la cohérence du graphe sémantique de tout schéma.

Le serveur peut maintenir plusieurs graphes sémantiques différents de plusieurs clients. Autrement dit, le même serveur est utilisé par différents clients, ces clients étant des éditeurs graphiques (EGOO, ...), des éditeurs textuels, ... Un graphe sémantique maintenu par le serveur peut être partagé par plusieurs clients ou n'être utilisé que par un seul client. Lorsqu'un graphe sémantique maintenu par le serveur est partagé par plusieurs clients, une modification du graphe sémantique (effacement d'un élément, insertion, modification, ...) doit être répercutée dans tous les clients le partageant. Par la suite, nous allons nous restreindre à un client particulier qui utilise le serveur, à savoir l'éditeur graphique.

Le serveur doit offrir à l'éditeur graphique les fonctionnalités minimales suivantes :

- Saisie des informations :

  Lors de la création d'un élément dans un schéma par l'éditeur graphique, celui-ci demande toutes les informations dont il a besoin au serveur. Ces informations sont de deux types :

  - Les chaînes de caractères : le serveur doit fournir à l'éditeur graphique la valeur des chaînes de caractères contenues éventuellement dans l'élément créé.

  - L'élément précis à insérer : le serveur peut déterminer quel est l'élément que l'éditeur graphique doit insérer dans le schéma . Nous verrons par la suite que ce type d'information, que le serveur doit fournir à l'éditeur graphique, ne concerne que l'insertion d'arcs. Cette notion sera détaillée dans le point suivant intitulé "Insertion d'un arc".

  Le serveur ne doit fournir à l'éditeur graphique que les deux types d'information cités plus haut. Mais il peut aussi demander plus d'information à l'utilisateur pour pouvoir maintenir sa propre information des éléments du graphe sémantique.

- Stockage des éléments du schéma :

  Le serveur doit stocker l'information, sous quelque forme que ce soit, relative aux éléments d'un graphe sémantique. Dans ce but, le serveur doit permettre de sauvegarder sur fichier ces informations.

- Demande d'information de la part de l'éditeur graphique :

  Le serveur doit pouvoir fournir toute l'information que l'utilisateur désire sur un élément du schéma représenté par l'éditeur graphique. Ces informations provenant du serveur ne sont pas des informations graphiques.

- Demande de suppression :

  L'éditeur graphique peut aussi demander un service au serveur lors de la suppression d'un élément du schéma par l'utilisateur. Dans ce cas, le serveur peut confirmer ou infirmer cette demande de suppression.

De plus, le serveur peut avoir l'initiative dans la communication avec le programme EGOO. Les fonctionnalités minimales décrites ci-dessus, que l'application graphique demande au serveur, peuvent être utilisées dans le sens Serveur vers EGOO (de manière plus restrictive car EGOO ne contient que l'information graphique d'un schéma). Autrement dit, le serveur peut signaler les différentes actions suivantes sur les éléments représentés :

- Suppression d'un élément :

  Lorsqu'un élément est supprimé dans le graphe sémantique maintenu par le serveur, celui-ci signale à l'éditeur graphique de le supprimer du schéma.

- Insertion d'un élément :

  Lorsqu'un élément est inséré dans le graphe sémantique du serveur, ce dernier le signale à l'éditeur graphique.

- Modification d'un élément :

  Lors d'une modification d'un élément du graphe sémantique du serveur affecte sa représentation graphique, le serveur le signale à l'éditeur graphique. Par exemple, si le schéma est un schéma E/A, la modification du nom d'un attribut d'un T.E. par le serveur entraîne que la représentation du T.E. doit être modifiée pour que le nouveau nom de l'attribut modifié soit affiché.

Le serveur et l'éditeur graphique sont donc deux programmes complètement indépendants qui interagissent lorsqu'une modification ou une demande d'information est opérée au niveau du serveur ou de l'éditeur graphique.

D'autres caractéristiques de la communication entre le serveur et le programme EGOO seront citées dans les points suivants.

2.4.5. Une fonctionnalité particulière : l'insertion d'un arc

Lors d'une insertion d'un arc sur un schéma, c'est-à-dire d'un élément de base reliant deux autres éléments de base, l'utilisateur doit désigner les deux éléments sur lesquels l'arc doit porter, ainsi que l'arc qu'il veut insérer. Ces trois éléments (l'arc et les deux éléments sur lesquels l'arc porte) doivent être communiqués au serveur par l'éditeur graphique. Le serveur, connaissant les éléments qui lui sont fournis, peut modifier l'arc qui doit être inséré entre les deux éléments. L'éditeur graphique doit donc pouvoir recevoir du serveur un arc à insérer différent de celui qui avait été prévu initialement.

En conclusion, dans le cas d'une insertion d'arc sur un schéma, le serveur peut donc modifier le type d'arc que l'utilisateur veut insérer.

Par exemple, prenons le cas du modèle MAG. Lors d'une insertion d'un chemin entre deux articles (le chemin étant un arc) l'éditeur graphique donne au serveur les deux articles sur lesquels porte le chemin ainsi que le chemin choisi par l'utilisateur. Le serveur retourne à l'éditeur graphique le chemin à insérer. Par exemple, si l'utilisateur choisit un chemin de classe fonctionnelle (1-1), alors que, d'après les informations maintenues par le serveur, ce chemin est de classe fonctionnelle (1-N), le serveur fournira à l'éditeur graphique l'arc représentant le chemin de classe fonctionnelle (1-N) et non (1-1).

De plus, il doit être possible de définir quels sont les éléments de base sur lesquels les arcs peuvent porter. Ainsi seul un certain nombre d'éléments de base sur lesquels un arc peut porter pourra être autorisé, ce qui limitera les erreurs lors de la création des schémas.

2.4.6. Création et manipulation d'éléments composés

Le logiciel EGOO doit pouvoir manipuler aussi bien des éléments composés que simples ou non composés. Nous pouvons définir deux types d'éléments. D'une part, les éléments qui sont, comme leur nom l'indique, composés et d'autre part, ceux qui ne le sont pas.

Un élément composé est un élément pouvant contenir d'autres éléments. Ces derniers peuvent être des éléments composés ou non composés. Un élément non composé est un élément ne pouvant contenir d'autres éléments.

Les éléments composés doivent êtres manipulés comme des éléments non composés, c'est-à-dire qu'il doit être possible de les insérer sur les schémas, les effacer, les déplacer, ... Mais il faut aussi fournir la possibilité d'insérer un élément dans un élément composé, de le supprimer, de le déplacer, ...

Nous illustrons cela avec l'exemple du modèle de la dynamique des traitements :

Les éléments non composés sont les messages, processus, ... et les éléments composés sont, par exemple, la délimitation entre messages externes et internes, ou encore le regroupement d'une partie du schéma qui forme un seul processus de niveau d'abstraction plus élevé. Un exemple est donné à la figure 14.

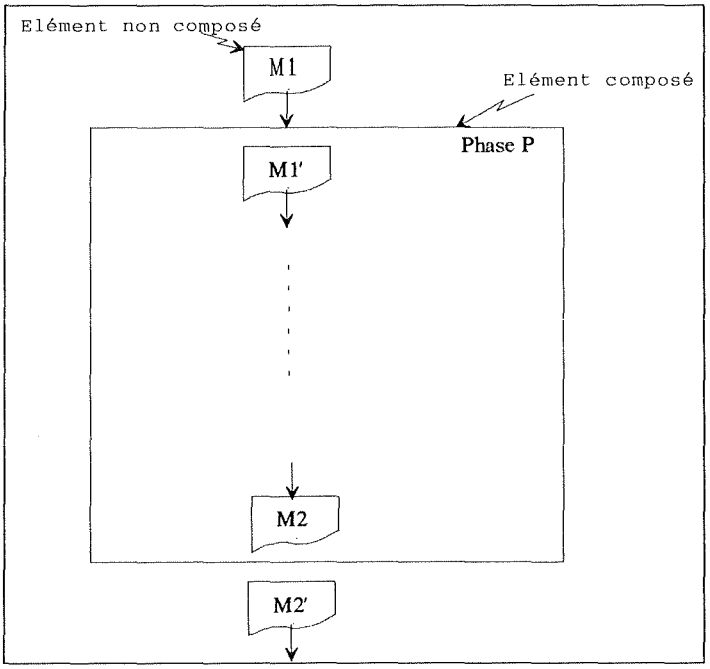

Fig. 14 : Elément composé

De plus, il doit être possible de paramétrer les déplacements d'éléments appartenant à des éléments composés. Autrement dit, le logiciel EGOO doit pouvoir alerter ou non le serveur lorsqu'un élément appartenant à un élément composé se déplace de l'intérieur de celui-ci vers l'extérieur, c'est-à-dire lorsqu'il sort du contour de l'élément composé.

Pour illustrer cette nécessité, prenons comme exemple le modèle du diagramme des flux. Les éléments composés sont les unités organisationnelles et les éléments non composés sont les messages, traitements, ... Lorsqu'un élément non composé est déplacé d'une unité organisationnelle à une autre, cela signifie qu'il change d'unité organisationnelle. Il doit donc être supprimé de la première unité organisationnelle et inséré dans la deuxième. Ceci doit se répercuter au niveau des informations maintenues par le serveur. Il faut, par conséquent, que le

programme EGOO avertisse le serveur de ce déplacement d'élément. Un exemple est donné à la figure 15.

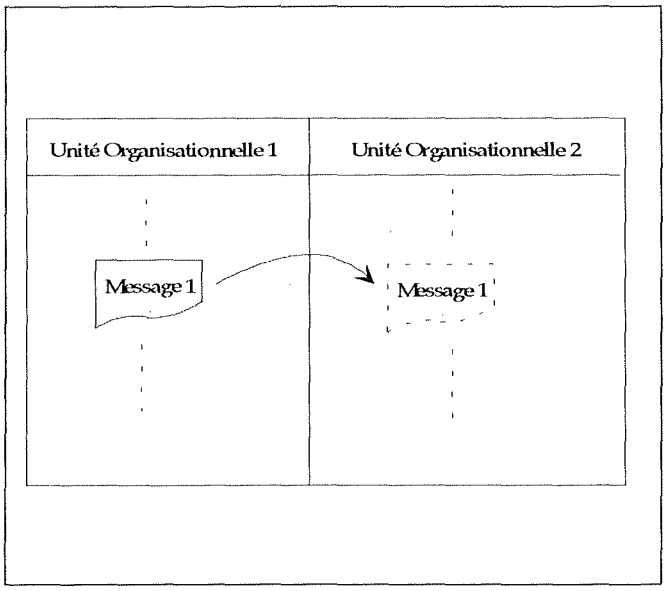

Fig. 15 : Déplacement d'un élément

2.4.7. Utilisation de sous-diagrammes

Vu la complexité souvent grande d'un schéma, nous devons pouvoir le représenter à des niveaux d'abstraction différents. Autrement dit, certaines parties d'un schéma peuvent être regroupées en un seul élément qui les représente (ces parties de schéma sont notées sous-diagrammes). Celui-ci est un élément appartenant à un niveau plus abstrait pour la représentation du schéma. Tout en étant un élément du schéma, cet élément ne peut pas être représenté sur le schéma représentant toutes les parties du schéma global, il doit donc être représenté sur un autre schéma (le lien entre cet élément et les éléments qu'il représente devant toujours exister).

Illustrons cette situation par l'exemple suivant :

Soit A1, A2, ..., An les éléments d'un schéma. Si les éléments B1,B2, ..., Bp forment un ensemble représentant un niveau d'abstraction plus élevé du schéma, alors le schéma est défini par B1, ... Bp où chaque Bi ($1 \leq i \leq p$) représente un ensemble de Ai. Pour représenter le schéma, nous aurons une fenêtre affichant les éléments B1, ... Bp et sur chaque élément Bi, nous devons pouvoir visualiser dans une autre fenêtre les éléments Aj qu'il représente. Cette caractéristique est illustrée à la figure 16.

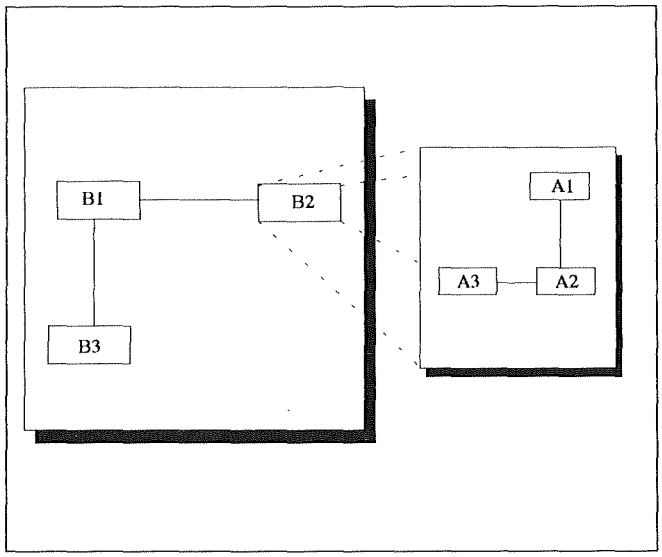

Fig. 16 : Sous-diagramme

Lorsque l'utilisateur désire afficher le sous-diagramme d'un élément du schéma, l'éditeur graphique demande au serveur le fichier contenant la description de ce schéma.

2.4.8. Multifenêtrage

Il doit être possible de fournir plusieurs vues, dans différentes fenêtres, d'un même schéma, ainsi que l'affichage de plusieurs schémas différents dans diverses fenêtres.

Pour un même schéma représenté dans différentes fenêtres, toute modification d'une partie du schéma dans une fenêtre doit être répercutée dans les autres fenêtres représentant ce même schéma.

2.4.9. Sauvetage des schémas

Le serveur s'occupant de toutes les informations non graphiques dans le schéma, l'éditeur graphique doit juste sauvegarder le graphisme d'un schéma. Autrement dit, l'éditeur graphique sauve toute l'information du schéma qu'il possède, excepté celle qu'il peut obtenir du serveur. Lors de la lecture du schéma sur disque, l'éditeur graphique doit questionner le serveur pour obtenir les informations manquantes des éléments qu'il doit représenter. De plus, s'il existe des éléments qui ont été ajoutés au schéma maintenu par le serveur, celui-ci avertit l'éditeur graphique pour l'insertion de ces nouveaux éléments sur le schéma représenté par ce dernier. De même, si certains éléments ont été supprimés ou modifiés, le serveur le signale à l'éditeur graphique.

Chapitre 3

3. Conception du logiciel

Ce chapitre fournit une description des principaux éléments de conception du logiciel EGOO. Une première partie exposera la conception générale du logiciel en décrivant les différents concepts déduits des schémas que le logiciel doit être capable de représenter : noeuds, arcs et droites complexes. Une seconde partie décrira le protocole de communication entre le logiciel EGOO et le serveur. Enfin, une troisième et dernière partie donnera une description succincte de l'approche orientée objet et des avantages qu'elle peut apporter à la structure du logiciel EGOO.

3.1. Conception générale

Dans les schémas qui doivent être représentés (E/A, MAG, ...), une caractéristique commune est à distinguer : nous pouvons considérer que tous les schémas cités se présentent sous forme de graphe. Autrement dit, ils sont tous composés de noeuds et d'arcs. Nous pouvons citer pour exemple le schéma E/A dont les noeuds sont les types d'associations (T.A.) et les types d'entités (T.E.), et dont les arcs sont les droites reliant les types d'associations aux types d'entités (rôles); de même pour le schéma MAG, les noeuds sont les types d'articles et items, et les arcs sont les segments de droites reliant ces types d'articles ou items.

Une remarque importante doit être formulée pour la suite du document : nous désignons les éléments du graphe par les noms Noeuds et Arcs. Cette notation est utilisée pour sa similarité avec les notions de la théorie des graphes, mais en aucun cas nous n'utiliserons cette théorie dans la conception du logiciel. En effet, dans les schémas qui peuvent êtres représentés par le logiciel, nous pouvons définir un arc portant, non pas uniquement sur deux noeuds, mais aussi sur deux arcs.

Pour chaque type de schéma à représenter, nous connaissons les types de noeuds et d'arcs qui vont être utilisés pour la construction d'un schéma particulier. Prenons par exemple un schéma simplifié E/A, nous aurons comme types de noeuds (illustrés à la figure 17) :

- T.E. : type de noeud qui décrit un type d'entité, composé d'un contour (un rectangle), d'un segment de droite dans la partie supérieure du rectangle, d'un titre (nom_du_TE) et d'un ensemble d'attributs (attribut_1, attribut_2, ...).

- TA : type de noeud qui décrit un type d'association, composé d'un contour (un polygone avec un segment de droite dans sa partie supérieure), d'un titre (nom_du_TA) et d'un ensemble d'attributs (attribut_1, attribut_2, ...).

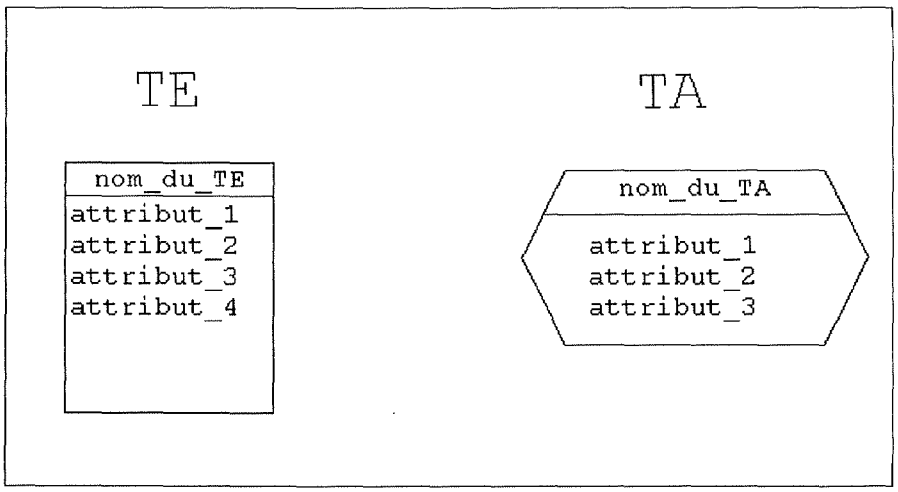

Fig. 17 : Types de noeuds

et comme types d'arcs (illustrés à la figure 18) :

- ArcSimple : arc composé uniquement d'un segment de droite, d'un rôle (role) et d'une contrainte de connectivité ((i-j)).

- ArcCI : arc composé d'un segment de droite, d'un cercle et d'une lettre "I" dans le cercle.

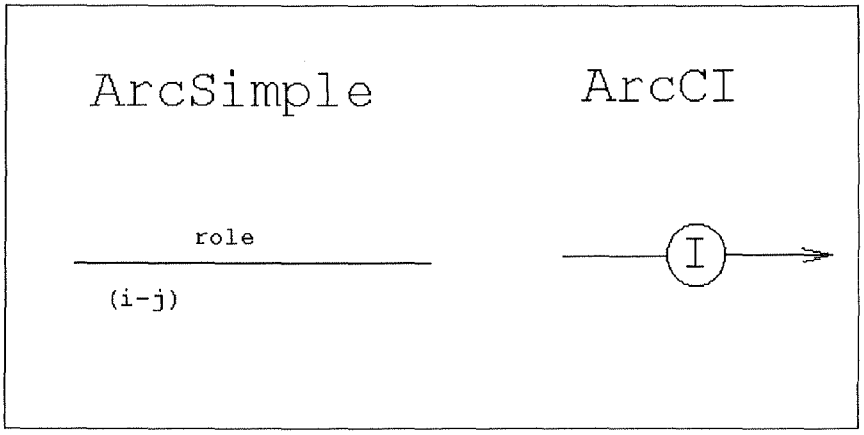

Fig. 18 : Types d'arcs

Avec ces deux types de noeuds et ces deux types d'arcs, nous pouvons construire une grande partie des schémas E/A (avec une description plus complète des types de noeuds et d'arcs, nous pourrions représenter tous les schémas E/A). Les noms nom_du_TE, nom_du_TA, attribut et role seront remplacés par les chaînes de caractères, qui, sur le schéma, donneront respectivement le nom du type d'entité, du type d'association, des attributs et le nom du rôle. Pour l'exemple de la figure 19, nous avons utilisé deux types de noeuds (T.E. et T.A.) et deux types d'arcs : ArcSimple et ArcCI. La construction d'un schéma peut donc se faire en utilisant

des types de noeuds et d'arcs définis à l'avance, en les particularisant aux noeuds et arcs voulus, et en les disposant les uns par rapport aux autres.

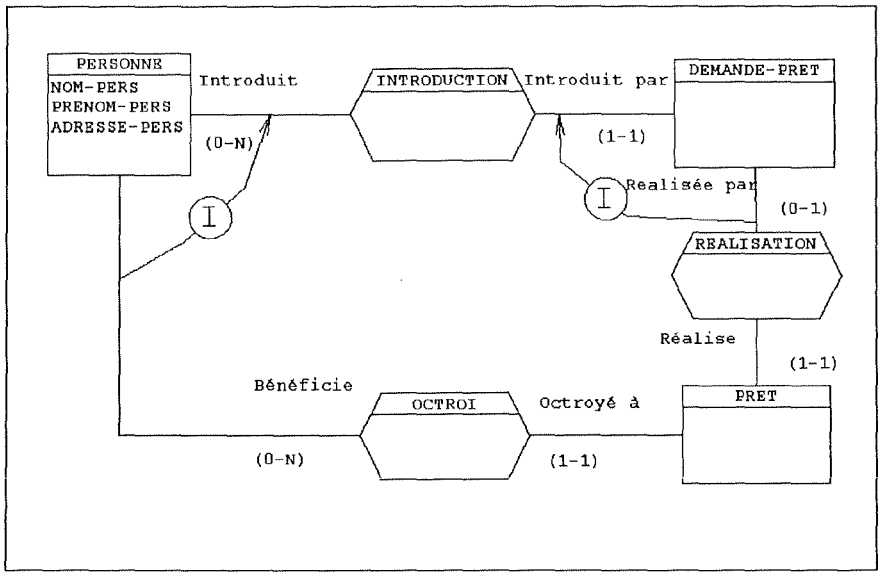

Fig. 19 : Schéma de type E/A

Remarque : Nous désignons par noeud et arc les occurrences des types de noeud et d'arc respectivement.

Dans les deux points suivants, nous allons décrire les concepts de noeud et d'arc des schémas. Le troisième point développé décrira le concept de droite complexe qui permet de définir une droite aux propriétés particulières.

Ces noeuds, arcs et droites complexes utilisent pour leurs représentations des éléments graphiques statiques et dynamiques. Donnons en leurs définitions.

Elément statique :

Un élément statique est un élément dont la valeur ne se modifie pas au cours du temps.

Elément dynamique :

Un élément dynamique est un élément dont la valeur doit être initialisée lors de sa création et qui peut être modifiée à n'importe quel moment de sa vie.

Les éléments statiques et dynamiques sont les suivants :

- Eléments statiques :
  - Droite : la droite peut être simple ou bien peut avoir une flèche à une de ses extrémités ou aux deux,
  - Rectangle,

- Texte statique : le texte statique est une chaîne de caractères qui ne peut varier dans le temps. Cette chaîne est définie une fois pour toute, lors de sa création,

- Polygone,

- Multi-droite : la multi-droite est un élément graphique composé d'un nombre quelconque de segments de droites mis bout à bout,

- Râteau : le râteau est un élément graphique composé de segments de droites placés les uns par rapport aux autres de façon à former un râteau (un exemple de râteau est illustré à la figure 20),

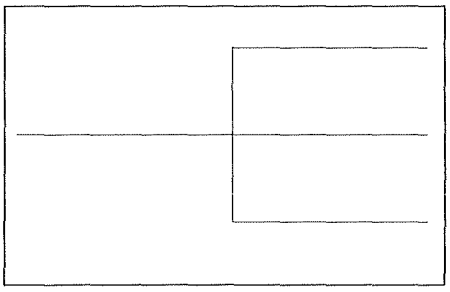

Fig. 20 : Râteau

- Ellipse.

- Eléments dynamiques :

  - Texte dynamique : le texte dynamique est une chaîne de caractères pouvant être modifiée,

  - Listebox : la listebox est une suite de chaînes de caractères mises les unes au-dessus des autres. Cette listebox peut être modifiée.

Illustrons par un exemple l'emploi de ces éléments dynamiques. Prenons pour cela, un T.E. du modèle E/A. Ce T.E. est composé d'un nom, représenté par un texte dynamique, et d'une liste d'attributs, représentée par une listebox. Ces deux éléments sont dynamiques car ils doivent être initialisés lors de la création du T.E., et ils peuvent être modifiés au cours de la vie du T.E..

3.1.1. Concept de noeud

Un noeud est un élément constitué des composants suivants :

- un contour,

- un ensemble de composants fixes,

- un ensemble de composants admissibles.

Décrivons à présent ces différents éléments.

Un noeud est un élément composé d'un contour défini par un ensemble d'éléments graphiques statiques (droites, rectangles, ...). Ces éléments forment une entité unique qui désigne de manière graphique le noeud. Par exemple, pour un type d'entité d'un schéma E/A, le contour du noeud définissant le type d'entité est constitué d'un rectangle. Le contour d'un noeud définissant un message dans un modèle de la dynamique des traitements est une suite de segments de droites et d'une courbe. Ces deux contours de noeuds sont illustrés respectivement à la figure 21.(a) et 21.(b).

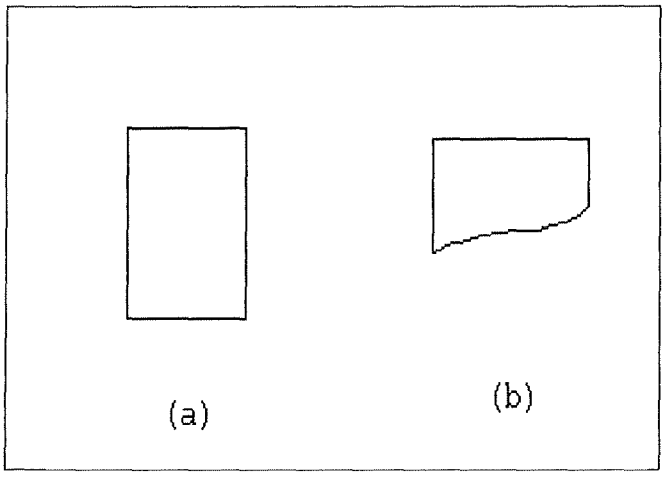

Fig. 21 : Contour de noeuds

Un noeud peut aussi être composé d'éléments fixes. Ces éléments fixes répondent à la définition suivante :

Elément fixe :

Un élément fixe est un élément graphique statique ou dynamique appartenant à un noeud ou à un arc, et existant pendant toute la durée de vie de ce noeud ou de cet arc. De plus, il ne peut être ni ajouté, ni supprimé du noeud ou de l'arc auquel il appartient et peut être manipulé indépendamment de ce noeud ou de cet arc (l'élément fixe est parfois noté composant fixe).

Par exemple, pour un noeud d'un type d'entité (T.E.) d'un schéma E/A, les éléments fixes sont le texte contenant le nom du T.E., la listebox contenant l'ensemble des attributs du T.E. ainsi que le segment de droite situé dans la partie supérieure du contour. Ces éléments, le nom du T.E., les attributs et le segment de droite existent pendant toute la durée de vie du noeud. Lorsque le T.E. est créé, le nom, les attributs et le segment de droite sont créés, et lorsque nous

supprimons le T.E., le nom, les attributs et le segment de droite sont supprimés. Ce noeud est illustré à la figure 22.

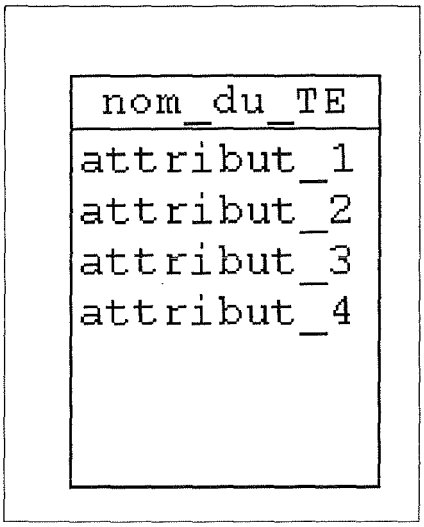

Fig. 22 : Composants fixes d'un noeud

De plus, nous pouvons définir des composants admissibles d'un noeud. Les composants admissibles répondent à la définition suivante :

Composant admissible :

Un composant admissible est un noeud ou un arc appartenant à un autre noeud. Tout composant admissible peut être ajouté, modifié ou supprimé indépendamment du noeud auquel il appartient.

Pour mieux comprendre ce concept de composants admissibles d'un noeud, nous allons l'illustrer par l'exemple suivant : le diagramme de flux illustré à la figure 23 est composé de "colonnes" que nous pouvons définir comme des noeuds (CLIENT, SERV. RECEPTION ET CONTROLES, ...). Ces noeuds contiennent des composants admissibles qui sont des messages, traitements, ... (BON COMM CLIENT, PRODUIT CLIENT, ...).

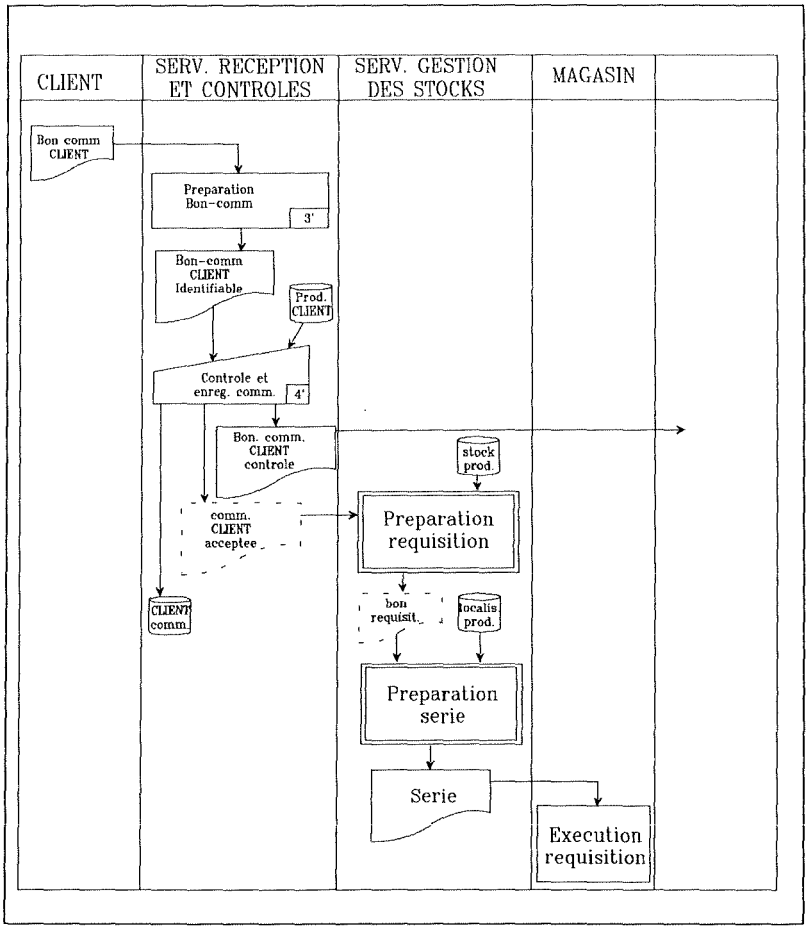

Fig. 23 : Diagramme de flux

Du point de vue graphique, il est possible de spécifier différentes contraintes sur un noeud ou sur certains de ses composants.

Dans un premier temps, voyons les contraintes que nous pouvons spécifier lorsque le noeud lui-même est sélectionné :

- Lors du déplacement d'un noeud (c'est-à-dire de son contour), l'ensemble des composants admissibles peut subir ou non ce déplacement.

- Lors du redimensionnement d'un noeud (c'est-à-dire de son contour), l'ensemble des composants admissibles peut subir ou non ce redimensionnement.

- Dans le cas d'un noeud qui est un composant admissible d'un autre noeud (ce dernier étant désigné comme le noeud père du premier noeud), il est possible de spécifier si son déplacement doit être limité ou non à l'intérieur du contour de son noeud père. Lorsque le noeud est déplacé de l'intérieur du contour de son noeud père dans le contour d'un autre

noeud, la possibilité d'alerter le serveur pour lui indiquer le changement de noeud père est offerte.

Etudions ensuite les contraintes qu'il est possible de spécifier lorsqu'un composant fixe d'un noeud est sélectionné.

- Le sens de redimensionnement du composant fixe peut être
  - horizontal : le composant fixe ne peut être redimensionné que dans le sens horizontal.
  - vertical : le composant fixe ne peut être redimensionné que dans le sens vertical.
  - vertical et horizontal : le composant fixe peut être redimensionné dans les deux sens (horizontal et vertical).
  - aucun : le composant fixe ne peut être redimensionné.
- Le composant fixe peut ou non subir un déplacement.

Certaines contraintes peuvent être ajoutées au composant fixe lorsque le noeud dont il fait partie est sélectionné et subit un déplacement ou un redimensionnement. Ces contraintes sont les suivantes :

- Lors d'un déplacement d'un noeud, le composant fixe peut, soit subir ce déplacement dans le sens horizontal, vertical, horizontal et vertical, soit ne pas le subir.
- Lors d'un redimensionnement du noeud, le composant fixe peut subir également un redimensionnement dans le sens vertical, horizontal, vertical et horizontal ou ne pas le subir.
- Lors d'un redimensionnement du noeud, il est possible de spécifier si le composant fixe est déplacé soit horizontalement, soit verticalement, soit encore dans les deux sens, par rapport à un certain point (ce point est noté point de contrainte) ou bien si, au contraire, il n'est pas déplacé par rapport à ce point (ce point est bien entendu déplacé pour suivre le redimensionnement effectué). Le non déplacement du composant fixe par rapport au point de contrainte, que ce soit horizontalement ou verticalement ou dans les deux sens, est réalisé de sorte que la distance horizontale ou verticale ou encore horizontale et verticale entre le point de contrainte et le coin supérieur gauche du composant fixe soit maintenue.

Prenons, par exemple, un noeud dont le contour est un rectangle et deux composants fixes qui sont aussi des rectangles, ces composants fixes étant, comme nous l'illustrons à la figure 24 à l'intérieur du contour.

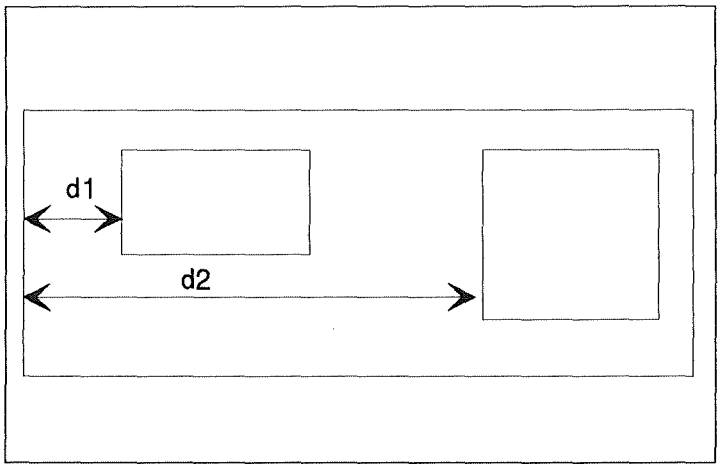

Fig. 24 : Redimensionnement sans contrainte

Supposons qu'il n'existe aucune contrainte sur les composants fixes. Autrement dit, supposons que le déplacement dû au redimensionnement et le redimensionnement des composants fixes soient autorisés dans les deux sens. La figure 25.(a) illustre le noeud avant redimensionnement. Lors du redimensionnement du noeud (Cf. figure 25.(b)), les composants fixes sont redimensionnés et les distances (d1 et d2) entre les composants fixes et le contour sont aussi redimensionnées de façon à ce que les proportions soient conservées. Autrement dit, les rapports des distances d1/d1' et d2/d2' sont égaux.

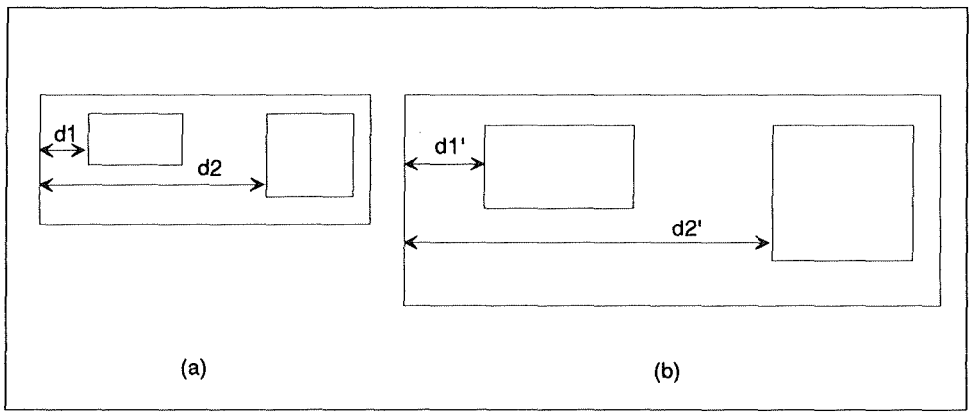

Fig. 25 : Après redimensionnement

Si par contre, les composants fixes ne peuvent pas être déplacés horizontalement par rapport au point de contrainte localisé au coin inférieur gauche du contour du noeud (ce point de contrainte se situe au point A de la figure 26.(a)), les distances horizontales entre le coin supérieur gauche des deux composants fixes et le point de contrainte sont conservées. Cette

situation est illustrée à la figure 26 où en 26.(a) le noeud est représenté avant redimensionnement et en 26.(b) après redimensionnement.

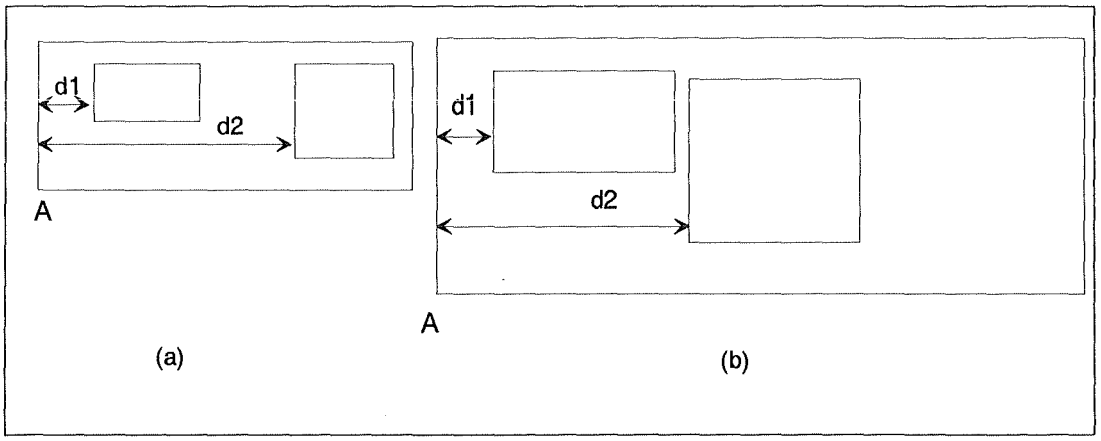

Fig. 26 : Redimensionnement avec contrainte

Pour mieux comprendre l'utilité de ces contraintes sur les éléments d'un noeud, nous allons spécifier différents noeuds de différents modèles.

Prenons le modèle E/A. Le noeud T.E. est défini de la manière suivante :

Le contour du noeud est un rectangle.

Les composants fixes sont :

- un texte qui représente le nom du T.E.. Ce composant fixe ne peut pas être redimensionné, ni être déplacé individuellement. Par contre, lors du redimensionnement du noeud, le composant fixe subit un redimensionnement horizontal et un déplacement. Lors d'un déplacement du noeud, cet élément fixe est également déplacé dans le sens vertical et horizontal. Ces contraintes permettent d'interdire le déplacement et le redimensionnement du nom du T.E. pour qu'il ne puisse se trouver en dehors du rectangle (ce qui aurait donner une mauvaise représentation du T.E.). Mais, lors d'un redimensionnement du noeud, le nom du T.E. doit être redimensionné horizontalement afin qu'il garde sa position par rapport au contour.

- une listebox qui représente les attributs du T.E.. Ce composant fixe ne peut être redimensionné individuellement que dans le sens vertical. Il ne peut être déplacé individuellement. Lors du redimensionnement du noeud, le composant fixe subit un redimensionnement horizontal et vertical, et ne subit aucun déplacement par rapport au coin supérieur gauche du contour. Lors du déplacement du noeud, l'élément fixe se déplace également dans le sens vertical et horizontal.

- un segment de droite situé dans la partie supérieure du contour en dessous du nom du T.E.. Le composant fixe ne peut être ni déplacé, ni redimensionné individuellement. Lors du redimensionnement du noeud, le composant fixe en question ne subit pas de déplacement par rapport au coin supérieur gauche du rectangle (point de contrainte) et ne subit le redimensionnement que dans le sens horizontal. Lors du déplacement du noeud, le composant fixe subit également ce déplacement dans le sens horizontal et vertical. Ces contraintes permettent donc au segment de droite de rester toujours à la même distance du bord supérieur du rectangle et d'avoir la même longueur que la largeur du rectangle.

La figure suivante illustre en 27.(a) le T.E. et en 27.(b) ce même T.E. après redimensionnement.

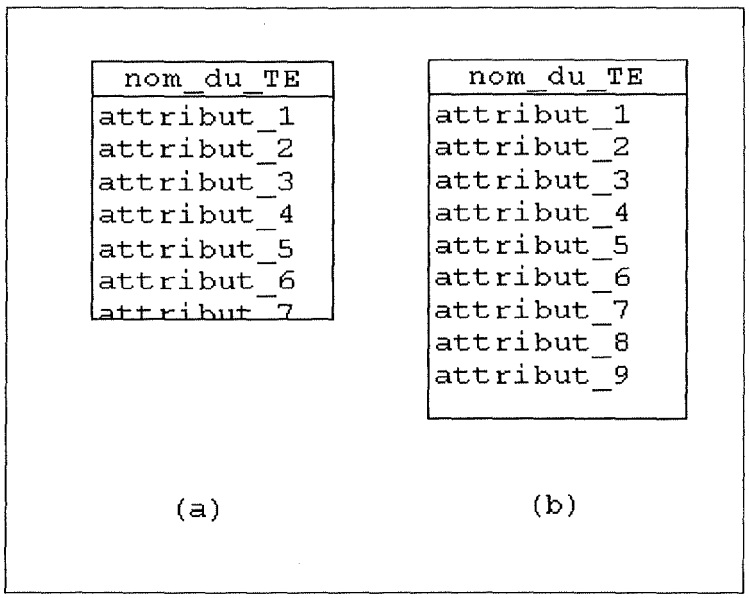

Fig. 27 : Redimensionnement d'un TE

Prenons comme second exemple le modèle du diagramme de flux. Le noeud représentant une tâche manuelle est défini de la manière suivante :

Le contour est un rectangle.

Les composants fixes sont :

- un texte qui représente le nom de la tâche manuelle. Ce composant fixe ne peut pas être redimensionné, ni être déplacé individuellement. Par contre, lors du redimensionnement du noeud, le composant fixe subit un redimensionnement horizontal ainsi qu'un déplacement. Lors du déplacement du noeud, le composant fixe subit également ce déplacement dans le sens horizontal et vertical.

- deux segments de droites placés dans le coin inférieur droit du rectangle. Ces composants fixes ne peuvent ni être redimensionnés, ni être déplacés individuellement. Lors du redimensionnement du noeud, ils ne peuvent pas subir le redimensionnement et ne peuvent pas non plus être déplacés horizontalement ou verticalement par rapport au point de contrainte situé dans le coin inférieur droit du contour. Lors du déplacement du noeud, les composants fixes subissent également ce déplacement dans le sens horizontal et vertical.

- un texte représentant le temps de réalisation de la tâche. Ce composant fixe possède les mêmes contraintes que les deux segments de droites définis ci-dessus.

La figure suivante illustre en 28.(a) la tâche manuelle et en 28.(b) cette même tâche après redimensionnement.

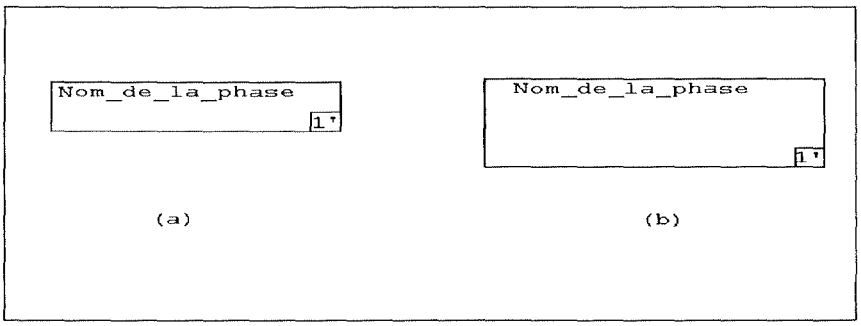

Fig. 28 : Redimensionnement d'une phase manuelle

De plus, un noeud peut posséder des points d'ancrage. Ces points sont les seuls endroits où les arcs peuvent venir se fixer sur le noeud. Si le noeud doit supporter des arcs, il doit donc au moins posséder un tel point.

Pour avertir le serveur de certaines actions réalisées sur le noeud, nous pouvons définir des noms de fonctions. Ceux-ci seront transmis au serveur lorsque les actions correspondantes seront réalisées. Les actions pour lesquelles il est possible de donner un nom de fonction, sont les suivantes :

- insérer : lorsqu'un nouveau noeud est créé et inséré dans un schéma.

- supprimer : lorsqu'un noeud est supprimé d'un schéma.

- modifier : lorsqu'un noeud est modifié sur un schéma, autrement dit, lorsque les valeurs des éléments dynamiques qu'il comprend sont modifiées.

- déplacer : lorsqu'un noeud est déplacé sur un schéma de son noeud père vers un autre noeud.

- naviguer : lorsqu'un noeud permet l'ouverture d'un sous-diagramme

- informer : lorsqu'un noeud demande les chaînes de caractères dont il est constitué.

Il est aussi possible de donner un nom de fonction pour l'action de modification réalisée sur des éléments fixes d'un noeud.

3.1.2. Concept d'arc

Un arc est un élément constitué des composants suivants :

- un contour

- un ensemble de composants fixes.

Un arc est un élément composé d'un contour défini par un seul élément graphique statique (droite, multi-droite ou râteau). Cet élément graphique désigne l'arc de manière unique. Par exemple, un arc d'un schéma E/A a pour contour une droite ou une multi-droite. Dans un schéma MAG, si un chemin relie un article à plusieurs autres articles, c'est-à-dire si le chemin est multi-cible, l'arc représentant le chemin a pour contour un râteau. Ces deux types de contour sont illustrés respectivement aux figures 29.(a) et 29.(b).

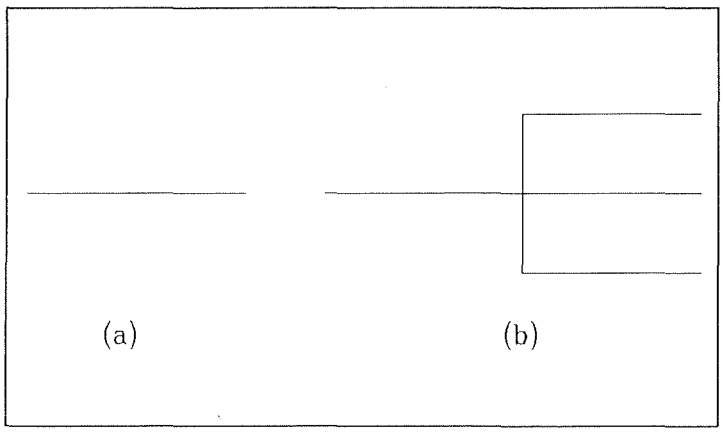

Fig. 29 : Contour d'arcs

L'arc peut aussi contenir des éléments fixes. Ces éléments fixes sont identiques à ceux que nous avons défini pour le concept de noeud. Par exemple, pour un arc représentant un rôle reliant un T.E. à un T.A. dans un modèle E/A les éléments fixes sont le nom du rôle et la connectivité. Cet arc est représenté à la figure 30.

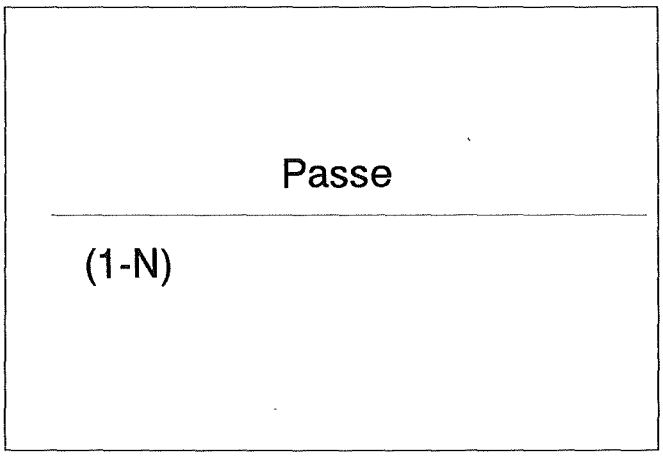

Fig. 30 : Arc représentant un rôle

Plusieurs contraintes peuvent être spécifiées pour les composants fixes de l'arc. Ces contraintes sont identiques à celles qu'il est possible de spécifier pour les composants fixes d'un noeud (voir p. 29).

De même, pour avertir le serveur de certaines actions réalisées sur l'arc, nous pouvons définir des noms de fonction. Ceux-ci seront transmis au serveur lorsque les actions correspondantes seront réalisées. Les actions pour lesquelles il est possible de donner un nom de fonction sont les suivantes : insérer, supprimer, modifier, informer (leur définition est identique à celle pour le noeud).

Il est aussi possible de donner un nom de fonction pour l'action de modification réalisée sur des éléments fixes de l'arc.

L'arc ayant comme contour un élément graphique statique, les extrémités de celui-ci seront localisées aux points d'ancrage des noeuds sur lesquels porte l'arc.

Notons pour terminer qu'il est également possible de spécifier l'ensemble des noeuds et arcs sur lesquels peut porter l'arc en question.

3.1.3. Le concept de droite complexe

Une droite complexe est composée d'une droite ou d'une multi-droite sur laquelle peuvent venir se disposer des éléments graphiques statiques obéissant à des contraintes de distance et de rotation.

Une droite complexe est donc un élément constitué des composants suivants :

- un contour (droite ou multi-droite),
- un ensemble d'éléments composés possédant des contraintes de distance et de rotation.

Ainsi définie, la droite complexe peut être assimilée à un élément graphique statique de base comme la droite ou la multi-droite (bien qu'à priori elle n'en soit pas un). Cette droite complexe peut donc être utilisée pour construire les noeuds et les arcs.

Deux points d'ancrage peuvent y être définis (ces points pouvant être distincts des extrémités du contour). Ils sont les lieux définissant les extrémités de la droite complexe. Pour illustrer la nécessité d'avoir des points d'ancrage distincts des extrémités du contour, considérons l'exemple du symbole montrant que deux chemins sont l'inverse l'un de l'autre dans le modèle MAG. Ce symbole est illustré à la figure 31.(a) où les points A et B représentent les extrémités de la droite complexe. Dans le cas où les points d'ancrage de la droite complexe sont les extrémités du contour de celle-ci, le symbole en question, représenté à la figure 31.(b), n'aurait pas la représentation souhaitée.

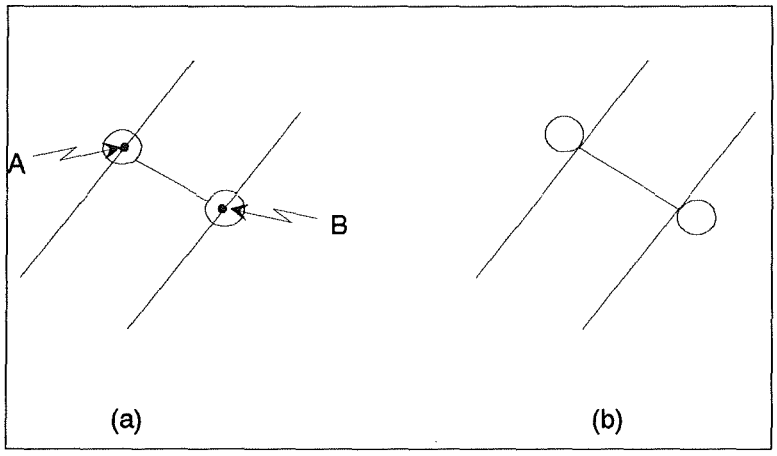

Fig. 31 : Points d'ancrage distincts du contour

Il est à remarquer que les éléments composés constituant la droite complexe (excepté le contour) ne varient pas en taille lors d'un redimensionnement de la droite complexe.

La droite complexe peut également être composée d'éléments composés possédant des contraintes de distance et de rotation. Définissons à présent ces éléments.

Elément composé d'une droite complexe :

Un élément composé d'une droite complexe est un ensemble d'éléments graphiques statiques formant un tout. Autrement dit, chaque élément composé est considéré comme un seul élément.

La contrainte de distance oblige l'élément composé à rester toujours à la même distance par rapport à un des deux points d'ancrage. La contrainte de rotation exprime si l'élément composé doit ou non subir une rotation lorsque la droite complexe est le siège d'une rotation.

Les contraintes de distance et de rotation sont maintenues comme suit :

Considérons une droite complexe ayant pour contour une multi-droite. Sur cette droite complexe, les deux points d'ancrage sont les extrémités du contour. A cette droite complexe, ajoutons un élément composé, défini par un segment de droite verticale proche de l'extrémité gauche du contour, cet élément composé respectant une contrainte de distance par rapport au point d'ancrage gauche et subissant la rotation de la droite complexe. Ajoutons également un élément composé défini par un ensemble de segments de droites formant un diabolo. Cet élément posséde une contrainte de rotation, mais pas de contrainte de distance. La droite complexe ainsi définie est illustrée à la figure 32.(a) (les points A et B désignent les extrémités de la droite complexe). Si la droite complexe doit être déplacée pour se trouver dans la situation de la droite en traits discontinus de la figure 32.(b), nous effectuerons les opérations suivantes :

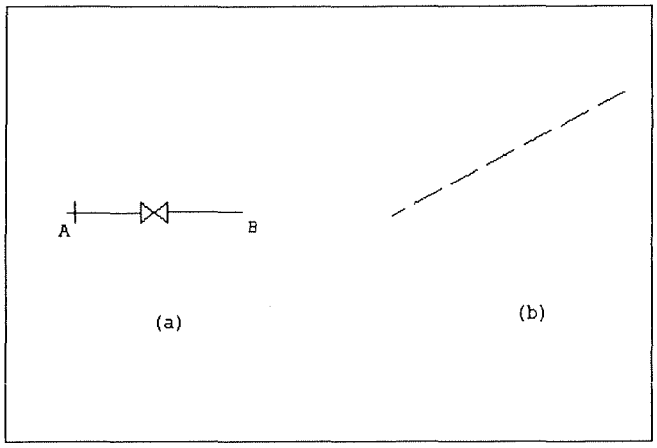

Fig. 32 : Droite complexe

Première phase : rotation

- rotation du contour de la droite complexe pour qu'elle soit dans la même direction que la droite en traits discontinus. La rotation est effectuée autour du point A.

- rotation des éléments composés. Cette rotation est obtenue en faisant tourner les extrémités des éléments composés autour du point A. Si un élément ne peut subir de rotation, il est simplement translaté de la distance séparant le point milieu du plus petit rectangle le contenant avant rotation et de ce même point milieu après rotation par rapport à A.

Cette première phase est illustrée à la figure 33 où en 33.(a) la droite complexe est représentée avant rotation, et en 33.(b) après avoir subi la rotation.

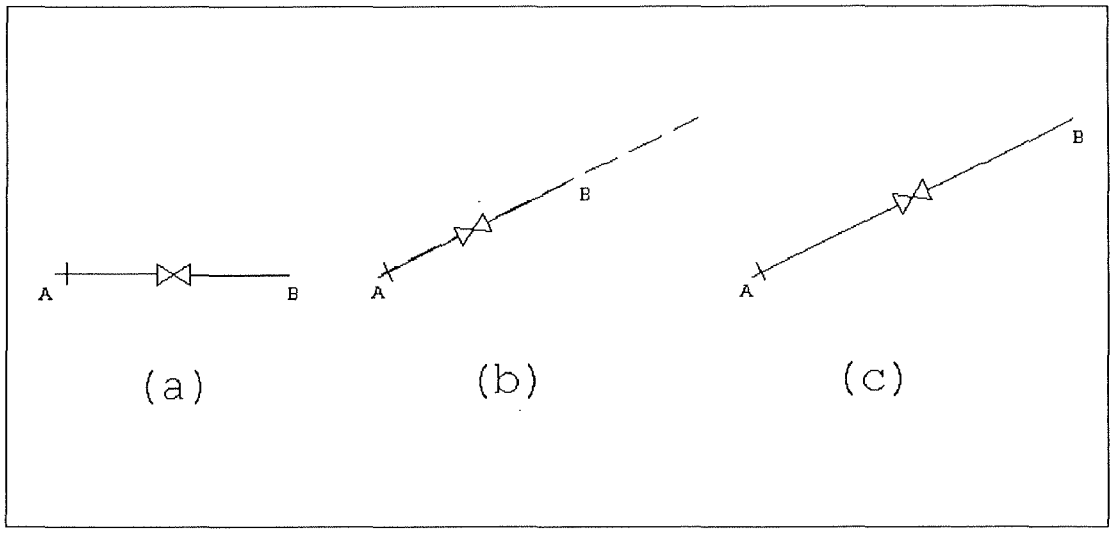

Fig. 33 : Rotation d'une droite complexe

Seconde phase : redimensionnement

- redimensionnement du contour pour qu'il corresponde à la droite en traits discontinus.

- déplacement des éléments composés avec contrainte de distance. Ce déplacement est tel que les distances horizontale et verticale par rapport au point d'ancrage sont conservées. Si un élément composé n'a pas de contrainte de distance, il est déplacé de sorte que le rapport de la longueur entre le point milieu du plus petit rectangle le contenant et le point A, et la longueur du contour est maintenu.

Cette seconde phase est illustrée à la figure 33.(c).

Nous allons à présent décrire le comportement des éléments composés lors de la création d'un nouveau segment de droite.

Reprenons le cas de la droite complexe étudiée précédemment. Cette droite est illustrée à la figure 34.(a). Si nous voulons que cette droite soit "cassée" en deux pour qu'elle puisse passer par le point C (illusté à la figure 34.(c)), nous procédons de la manière suivante :

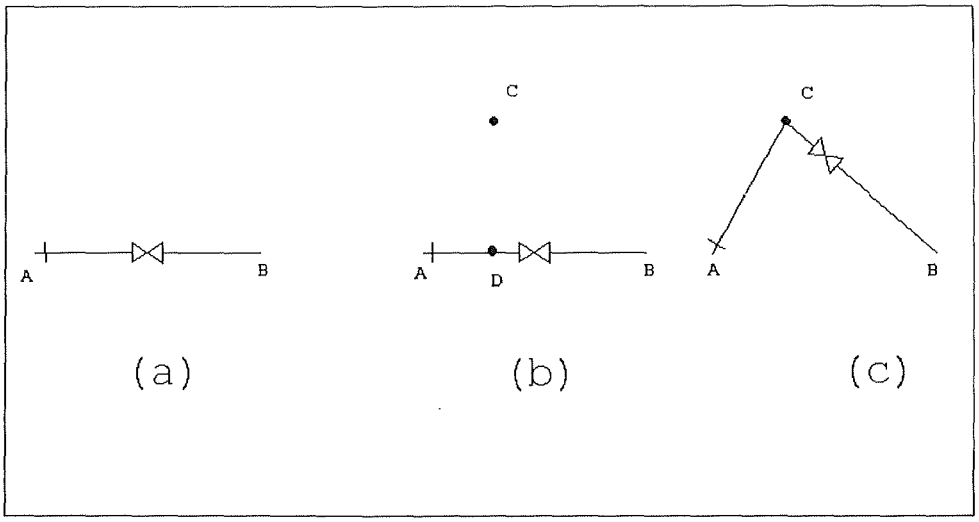

Fig. 34 : " Cassure " d'une droite complexe

- ajout d'un point D sur la droite AB. Ce point D est le point d'intersection de la droite AB et de la droite perpendiculaire à AB passant par le point C. Nous n'avons donc plus un seul segment de droite, mais deux segments de droite (le segment AD et le segment DB). Cette situation est illustrée à la figure 34.(b).

- déplacement du point D pour qu'il soit confondu avec le point C. Ce déplacement est réalisé en effectuant une rotation autour du point A du segment de droite AD ainsi qu'un déplacement de ce même segment pour qu'il se positionne sur le segment AC. Cette opération est effectuée de la même manière que pour une droite complexe dont le contour est une multi-droite composée d'un seul segment (ce cas a été étudié précedemment). Nous procédons de manière identique avec le segment DB : le segment DB subit une rotation autour du point B et un redimensionnement pour qu'il soit confondu avec le segment CB. Nous illustrons cette situation à la figure 34.(c).

Les éléments composés du segment de droite AB se situent à présent sur le segment de droite AC s'ils étaient sur le segment de droite AD et ils sont sur CB s'ils étaient avant sur DB.

Supposons à présent que nous voulions supprimer un segment de droite d'une droite complexe. Considérons par exemple la droite complexe illustrée à la figure 34.(c). Dans le but de ne garder qu'un seul segment de cette droite complexe au lieu des deux segments originaux (AC et CB), nous choisissons de supprimer le point intermédiaire C. La droite complexe que nous voulons obtenir est bien entendu celle qui est illustrée à la figure 34.(a). Pour arriver à ce résultat, nous procédons de la manière suivante :

- Dans un premier temps, nous définissons le point D, intersection de la droite AB et de la droite perpendiculaire à AB passant par C.

- Ensuite, nous faisons effectuer au segment AC une rotation de centre A suivie d'un redimensionnement de sorte que le segment en question se positionne sur le segment AD.

- De la même manière, nous faisons effectuer au segment BC une rotation de centre B suivie d'un redimensionnement de sorte que le segment en question se positionne sur le segment BD.

- Enfin, nous ne tenons plus compte du point D, qui est supprimé.

3.1.4. Conception de l'éditeur graphique

Nous allons décrire dans cette partie l'architecture générale ainsi que les relations entre les différents composants de l'architecture de l'éditeur graphique.

L'éditeur graphique contient à un moment donné un ensemble de schémas qui ont été créés par l'utilisateur. Pour chacun de ces schémas, plusieurs fenêtres, indépendantes quant à la vue qu'elles offrent sur le schéma, peuvent être ouvertes.

Ces fenêtres sur les schémas offrent les fonctionnalités de zoom, de modification de la vue du schéma, etc ....

Chaque schéma contient deux ensembles de noeuds et d'arcs.

Un premier ensemble contient tous les noeuds et arcs définis par l'utilisateur qui servent de "briques de base" pour l'élaboration de la représentation graphique du schéma. Ce premier ensemble est appelé les ressources. Ces ressources contiennent donc les noeuds et arcs définis dans les points 3.1.1. et 3.1.2. (respectivement intitulés "Concept de noeud" et "Concept d'arc"). Les droites complexes définies au point 3.1.3. intitulé "Concept de droite complexe" sont également contenues dans les ressources et sont donc utilisées comme éléments graphiques statiques de base pour la définition des noeuds et arcs.

Un deuxième ensemble contient les noeuds et arcs que l'utilisateur a inséré sur le schéma graphique. Nous appellerons ce deuxième ensemble le dessin. Celui-ci est affiché dans la fenêtre du schéma et représente ce que l'utilisateur construit.

Lors de la création d'un schéma par l'utilisateur, la communication entre ce schéma et le serveur est initialisée. Ce dernier crée un graphe sémantique correspondant au schéma de l'éditeur graphique.

Lorsque l'utilisateur veut insérer un noeud sur le dessin, il choisit un noeud dans les ressources et demande à l'éditeur graphique de l'insérer sur le dessin. A ce moment, l'éditeur graphique appelle le serveur pour que celui-ci lui fournisse toutes les informations nécessaires pour l'initialisation des éléments graphiques dynamiques qui sont contenus dans le noeud.

L'éditeur graphique recevant ces informations (I.e. les chaînes de caractères), peut dupliquer le noeud des ressources et l'insérer avec les éléments graphiques dynamiques du noeud initialisé.

Chaque noeud du dessin contient toujours une référence au noeud des ressources dont il est la "copie".

Toute action réalisée sur un noeud du dessin qui correspond à une fonction définie dans les ressources du noeud est transmise au serveur. Ensuite, suivant le résultat fourni par le serveur, l'éditeur graphique exécute ou non l'action demandée par l'utilisateur.

Notons pour terminer que lors de l'insertion d'un arc entre deux noeuds ou deux arcs, la procédure est parfaitement identique à la procédure effectuée lors de l'insertion d'un noeud.

3.2. Communication EGOO - Serveur

Cette partie est consacrée à la conception de la communication entre le serveur et l'éditeur graphique EGOO.

Pour rappel, l'éditeur graphique ne maintient que l'information relative à la représentation et à la disposition des éléments contenus dans le schéma qu'il représente. Une application, appelée serveur, maintient pour sa part le graphe sémantique du schéma.

Nous allons rappeler brièvement les diverses fonctionnalités demandées au serveur (ces fonctionnalités ont été décrites dans le point 2.4.4. intitulé " Utilisation d'un serveur " ainsi que dans les points suivants).

- Saisie des informations,
- Stockage des éléments du schéma,
- Demande d'information de la part de l'éditeur graphique,
- Demande de suppression,
- Sous-diagrammes,
- Déplacement d'un noeud ou d'un arc d'un noeud père vers un autre noeud père.

Le serveur peut avoir l'initiative dans la communication avec l'éditeur graphique. Dès lors, les fonctionnalités suivantes sont également demandées au serveur:

- Suppression d'un élément,

- Insertion d'un élément,

- Modification d'un élément.

La communication entre l'éditeur graphique et le serveur est réglée de la manière suivante. Lorsque l'éditeur graphique (respectivement le serveur) a besoin d'appeler le serveur (respectivement l'éditeur graphique), il se met en attente jusqu'à ce qu'il ait reçu la réponse à son appel. Autrement dit, la communication entre l'éditeur graphique et le serveur est une communication synchrone. Ce type de communication a été choisi pour ne pas compliquer la tâche lors de la conception et de l'implantation de l'éditeur graphique.

Etant donné que le serveur et l'éditeur graphique maintiennent des informations sur les mêmes noeuds et arcs d'un schéma et qu'ils doivent se communiquer des informations à propos de certains de ces noeuds et arcs, ceux-ci doivent avoir un identifiant unique. Le serveur est chargé de donner un identifiant aux noeuds ou aux arcs lors de leurs créations. Cet identifiant est utilisé pour désigner le noeud ou l'arc qui lui correspond dans la communication entre EGOO et le serveur.

Dans la suite, nous allons exposer les différentes requêtes que l'éditeur graphique et le serveur peuvent formuler.

Nous allons tout d'abord commencer par les requêtes que l'éditeur graphique peut formuler au serveur.

Initialiser :

Requête pour initialiser la communication entre l'éditeur graphique et le serveur. Les paramètres fournis au serveur sont le type de schéma qui doit être représenté (E/A, MAG, ...) et le nom du fichier contenant le schéma. Ce dernier est fourni au serveur pour qu'il puisse enregistrer, pour chacun de ses graphes sémantiques, le fichier de l'éditeur graphique qui correspond.

Insérer :

Requête qui signale au serveur que l'utilisateur veut insérer un noeud ou un arc sur le schéma.

Supprimer :

Requête qui signale au serveur que l'utilisateur veut supprimer un noeud ou un arc sur le schéma.

Modifier :

Requête qui signale au serveur que l'utilisateur veut modifier un noeud, un arc ou un élément fixe d'un noeud ou d'un arc sur le schéma. Autrement dit, l'utilisateur veut

modifier les chaînes de caractères que contient le noeud, l'arc ou l'élément fixe en question.

Déplacer :

Requête qui signale au serveur que l'utilisateur veut déplacer un noeud ou un arc sur le schéma. Cette requête est appelée par l'éditeur graphique lorsqu'un noeud est déplacé de l'intérieur du contour de son noeud père vers l'intérieur du contour d'un autre noeud.

Dupliquer :

Requête qui signale au serveur que l'utilisateur veut dupliquer un noeud ou un arc sur le schéma. L'élément dupliqué a bien entendu un identifiant différent de l'identifiant de l'élément qui est à la base de la duplication.

Naviguer :

Requête qui signale au serveur que l'utilisateur veut afficher le sous-diagramme qui est représenté par un noeud du schéma.

Informer :

Requête qui demande au serveur de fournir les chaînes de caractères contenues dans un noeud ou arc du schéma.

Terminer :

Requête pour terminer la communication entre le serveur et l'éditeur graphique au sujet du schéma sur lequel ils travaillent.

A chacune des requêtes que l'éditeur graphique soumet au serveur, la réponse de ce dernier peut être positive ou négative. Si le serveur sait satisfaire la requête demandée, il fournit à l'éditeur graphique les résultats de cette requête (par exemple, l'identifiant de l'élément à insérer, la confirmation de la suppression d'un élément, etc ...). Si le serveur ne peut satisfaire la requête, il en fait part à l'éditeur graphique.

Nous allons à présent exposer les requêtes que le serveur peut formuler à l'éditeur graphique. Il est à remarquer que le serveur ne peut demander des requêtes à l'éditeur graphique que si ce dernier a initialisé la communication.

Les requêtes que le serveur peut demander à l'éditeur graphique sont les suivantes.

- Insérer
- Supprimer
- Modifier

- Déplacer.

Ces requêtes ont une signification identique à celles décrites précédemment.

3.3. Conception et programmation orientées objet

3.3.1. Bases de l'orienté objet

La programmation orientée objet est basée sur les concepts d'objet, de classe, d'héritage de classes, de polymorphisme et enfin, de lien dynamique entre appel de procédure et code. Nous allons développer chacun de ces concepts.

Les objets

Dans un système orienté objet, l'élément de base est l'objet. Il est défini d'une part par son état à un moment donné et d'autre part, par l'ensemble de ses méthodes, c'est-à-dire l'ensemble des procédures et fonctions définissant les opérations significatives à réaliser sur l'objet en question. La définition d'un objet regroupe donc la notion d'état et la notion de comportement. La capacité de réunir à la fois des données (l'état) et du code (le comportement) dans un même objet est généralement appelée " l'encapsulation ".

Les classes

La classe est définie par l'ensemble des objets possibles. Pour plus de clarté, comparons les notions de classe et d'objet aux notions de type et de variable du langage C : les classes d'objets correspondent bien entendu aux types de variables tandis que les objets, instances des classes, peuvent être mis en relation avec les variables, "instances" des types de variables.

Il est à noter qu'une classe possède une partie visible et une partie cachée. Ces deux parties peuvent contenir des méthodes ou des éléments qui définiront les états des objets. Les éléments composant la partie cachée ne peuvent être accessibles que par les méthodes des objets de la classe en question, tandis que les éléments constituant la partie visible sont accessibles à n'importe quelle méthode de n'importe quel objet.

L'héritage des classes

Les définition et implantation d'une classe peuvent se baser sur celles de classes déjà existantes. Ce principe est à la base de la notion d'héritage des classes. En effet, prenons deux classes notées X et Y. Si, comme nous l'avons indiqué sur la figure 35, la classe Y hérite de la classe X, nous appellerons Y la classe dérivée et X la classe de base. La classe Y possède alors

d'une part une partie dérivée de la classe X et d'autre part, une partie supplémentaire écrite spécialement pour elle.

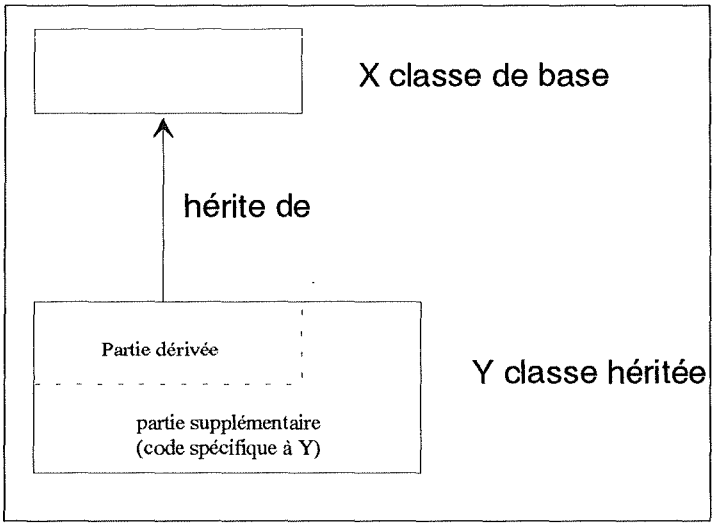

Fig. 35 : Héritage des classes

Notons que la partie dérivée de Y n'est pas nécessairement identique à la classe de base X. En effet, la correspondance relative à l'héritage des classes n'est pas toujours une identité mais peut parfois être beaucoup plus riche. Il peut y avoir différentes transformations entre la classe de base et la partie dérivée (par exemple : une caractéristique de la classe X peut être renommée, réimplantée, dupliquée, ...).

Remarquons qu'étant donné son fonctionnement, le principe d'héritage des classes minimise la quantité de nouveau code à fournir quand le programmeur utilise des classes plus spécifiques que des classes existantes, c'est-à-dire ayant des caractéristiques supplémentaires. Autrement dit, lors de l'extension d'un système existant, la notion d'héritage des classes peut s'avérer très utile et très économique. De la même façon, nous pouvons dire que le principe d'héritage des classes facilite la réutilisation du code existant.

Généralement, la relation d'héritage des classes est appelée relation " est un " (plus connue encore sous le nom de relation " is a "). Le nom de cette relation provient du fait suivant : lorsque la classe Y hérite de la classe X, la classe Y possède toutes les caractéristiques de X et donc, " est un " X. Dans l'exemple suivant, nous avons défini la classe de base Elément géométrique et trois classes dérivées Droite, Polygone et Ellipse. Dès lors, Droite " est un " Elément géométrique. Il en est de même pour Polygone ou Ellipse. La relation d'héritage de ces différentes classes est illustrée à la figure 36.

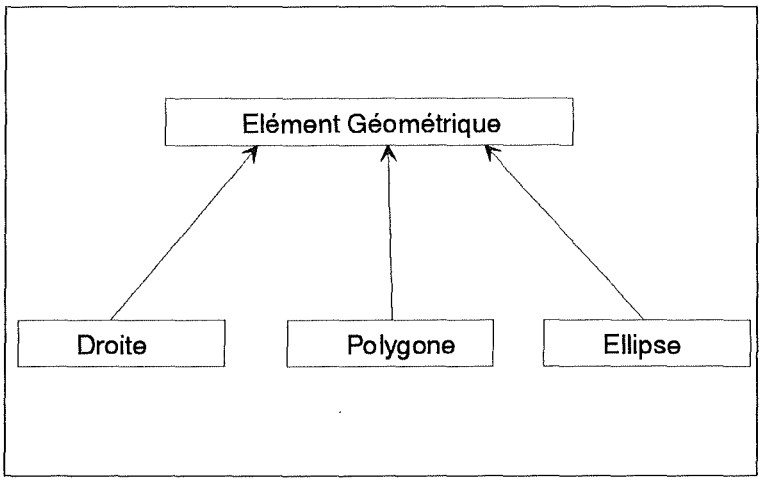

Fig. 36 : Exemple d'héritage des classes

Le polymorphisme

Le polymorphisme est la capacité qu'ont les objets de passer les uns pour les autres. Dans le même contexte, nous appellerons référence polymorphe une référence qui, au cours du temps, référence des instances de plus d'une classe. Une telle référence dispose d'une part d'un type statique et d'autre part, d'un type dynamique variant au cours de l'exécution du programme. Il est à noter que l'ensemble des valeurs que peut prendre le type dynamique d'une référence polymorphe, est l'ensemble des types statiques des classes dérivées de la classe référencée (il est bien entendu que le type dynamique peut également être égal au type statique de la classe de référence).

La relation " est un " définie dans le point précédent, relatif à l'héritage des classes, est fortement liée au concept de polymorphisme. En effet, si la classe Y est dérivée de la classe X, Y est un X (comme nous l'avons déjà vu) et de plus, à tout moment, lorsqu'une instance de la classe X est attendue, une instance de la classe Y est permise. En guise d'illustration, considérons l'exemple suivant:

Soient X et Y deux classes telles que la classe Y soit dérivée de la classe X. Considérons le pseudo-programme suivant :

```
x : X
y : Y
x.Tracer        (1)
y.Tracer        (2)
x := y          (3)
x.Tracer        (4)
```

Après l'exécution de la ligne (1), le type statique de la variable x est X. Il en est de même pour le type dynamique. Par contre, après l'exécution de la ligne (3), le type statique de la variable x est toujours X, mais son type dynamique est maintenant Y.

Lien dynamique entre appel de procédure et code

Le lien dynamique auquel nous faisons référence est le lien entre l'appel de procédure et le code qui sera exécuté en réponse à l'appel. Ce lien est dit dynamique car le code associé à l'appel en question n'est connu qu'au moment de l'exécution. Il est à noter que cette notion de lien dynamique entre appel de procédure et code est indissociable des notions de polymorphisme et d'héritage des classes.

Décrivons à présent ce qu'est véritablement ce lien dynamique.

Supposons que x soit une référence polymorphe dont le type statique est X et dont le type dynamique est soit X, soit Y. Supposons, de plus, que la procédure appelée soit définie dans la classe X et redéfinie dans la classe Y comme une partie dérivée de X. L'appel de cette procédure dépend du type dynamique de x. S'il s'agit de X, l'appel sera relié au code défini dans la classe X. Par contre, s'il s'agit de Y, l'appel sera relié au code redéfini dans la classe Y.

Reprenons pour exemple le schéma précédent et supposons que la classe Elément géométrique contienne la procédure Dessiner (dont l'implantation peut très bien être vide). Suite à l'héritage des classes, les classes Droite, Polygone, Ellipse ont également une procédure Dessiner. Cette procédure peut être redéfinie dans chaque classe. Si, par exemple, nous disposons d'un tableau d'éléments géométriques, il suffit, pour dessiner l'ensemble de ces éléments, d'écrire la procédure suivante :

```
Pour i allant de 1 jusqu'au nombre d'éléments géométriques

    Dessiner le ième élément géométrique.
```

A chaque passage dans la boucle, le code véritablement appelé sera celui qui correspond au type dynamique de la référence polymorphe en question. L'énorme avantage que procure ce type de travail est le suivant : si une classe dérivée de la classe Elément géométrique doit être ajoutée lors d'une extension du système, la procédure reste totalement inchangée. Cette simplicité de réalisation est à opposer à la complexité des traitements par cas traditionnels. En effet, un traitement par cas aurait mené à la considération d'une procédure du style :

```
Pour i allant de 1 jusqu'au nombre d'éléments géométriques

    si l'élément est une droite

    alors Dessiner_Droite

    sinon si l'élément est ...
```

Dans ce cas, lors d'une adjonction d'une nouvelle classe dérivée de la classe Elément Géométrique, la procédure aurait dû être modifiée pour traiter ce nouvel élément géométrique. Cela constitue un des désavantages majeurs de la programmation traditionnelle auquel remédie la programmation orientée objet.

3.3.2. Utilisation de l'orienté objet dans le logiciel EGOO

Au niveau de la conception du logiciel, l'orienté objet permet de structurer les différents composants qui interviendront dans l'architecture du logiciel, ainsi que les liens existant entre ces différents composants. Les différents composants qu'il est possible de mettre en évidence dans l'architecture du logiciel EGOO, sont le schéma, la fenêtre, le noeud, l'arc, les éléments graphiques (droite, rectangle, ...), ... Nous pouvons voir ces éléments comme des objets. Ils possèdent chacun un état (des données) et des méthodes associées fournissant différents services sur l'objet.

Pour mieux comprendre la vision orientée objet que nous pouvons avoir sur ces différents composants, nous allons illustrer les données et les méthodes associées aux objets schéma et noeud utilisés. Pour une description détaillée de tous les objets (données et méthodes) utilisés par le logiciel EGOO nous renvoyons le lecteur au chapitre suivant.

Prenons l'objet schéma, comme premier objet. Cet objet est composé des données suivantes :

- la liste des fenêtres qui offrent une vue sur la représentation graphique d'un schéma (ces fenêtres sont elles-mêmes des objets),

- la liste des noeuds et arcs qui composent la représentation graphique d'un schéma.

Les méthodes associées à l'objet schéma sont les suivants :

- ajout d'une nouvelle fenêtre dans la liste des fenêtres,

- suppression d'une fenêtre dans la liste des fenêtres,

- ajout d'un nouveau noeud (ou arc) sur la représentation graphique du schéma,

- suppression d'un noeud (ou arc) sur la représentation graphique du schéma.

Illustrons à présent les données et les méthodes définissant l'objet noeud. L'objet noeud a pour données :

- la liste des éléments graphiques qui représentent le noeud,

- la liste de références d'arcs qui portent sur le noeud.

Les méthodes associées sont les suivantes :

- ajout ou suppression d'une référence d'arc portant sur le noeud,
- tracé du noeud sur la représentation graphique,
- déplacement du noeud sur la représentation graphique,
- ...

Les différents éléments intervenant dans l'architecture du logiciel, cités au point 3.1.4. (intitulé " Conception de l'éditeur graphique "), peuvent donc être vus comme des objets. Le logiciel est donc structuré en une collection d'objets. Ces objets ont leurs propres comportements et interagissent avec les autres objets de l'architecture du logiciel. Dans le chapitre suivant, nous décrirons les classes de ces objets.

Chapitre 4

4. Implantation

Ce chapitre est consacré à la partie implantation du logiciel EGOO. Nous allons y décrire les principales techniques utilisées pour implanter les différents concepts décrits dans le chapitre 3 intitulé " Conception du logiciel ".

Dans un premier temps, nous étudierons l'architecture globale du logiciel EGOO.

Ensuite, nous décrirons le fichier de ressources qui est utilisé pour contenir toute la description des noeuds, arcs et droites complexes qui servent d'éléments de base pour la création des dessins des schémas.

Nous poursuiverons en exposant le principe de communication de MS-Windows utilisé par le logiciel pour communiquer avec le serveur. Cette partie décrira également le type d'information échangée entre le serveur et l'éditeur graphique.

La quatrième partie sera consacrée à l'implantation du multi-fenêtrage utilisé par le logiciel.

Nous développerons alors les classes d'objets utilisés pour définir les éléments constituant l'architecture du logiciel.

Enfin, nous donnerons quelques techniques utilisées pour l'inplantation du logiciel.

4.1. Architecture du logiciel

Les différents éléments intervenant dans l'architecture de l'éditeur graphique sont les suivants :

- les classes d'objets,
- le fichier de ressources,
- le serveur,
- la fonction principale de l'éditeur graphique,
- l'environnement MS-Windows.

Nous allons par la suite décrire chacun de ces différents éléments ainsi que leurs relations.

Tout programme réalisé dans l'environnement MS-Windows doit posséder une fonction (que nous noterons fonction principale) déclarée comme étant la fonction recevant tous les messages de l'environnement destinés au programme. Autrement dit, chaque fois que l'environnement doit signaler un événement au programme, il appelle la fonction principale de ce programme en lui fournissant un message décrivant cet événement.

Les types de messages que la fonction principale peut recevoir sont les suivants :

- sélection d'un item d'un menu,
- déplacement de la souris,
- enfoncement du bouton gauche de la souris,
- enfoncement d'une touche du clavier,
- ...

La fonction principale de l'éditeur graphique connaît les différents schémas qui existent à un moment donné de l'exécution du programme. Chaque schéma est un objet qui possède une liste d'objets représentant des noeuds et des arcs. Cette liste représente le dessin du schéma.

Lorsqu'un message est envoyé à la fonction principale, celle-ci le traite en appelant différentes méthodes d'objets noeuds et arcs du schéma ou du schéma lui-même.

Pour illustrer cette situation, prenons par exemple la création d'un noeud T.E. sur un schéma E/A. L'utilisateur choisit, dans le menu, l'item correspondant à la création d'un noeud. A ce moment, la fonction principale de l'éditeur graphique reçoit un message signalant que l'item de création d'un noeud a été sélectionné. Ce message est traité, en demandant tout d'abord à l'utilisateur quel est le noeud qu'il veut créer (T.E. ou T.A.). Suite à sa réponse, un objet noeud est créé et une méthode de l'objet schéma courant est appelée pour ajouter à son dessin le nouveau noeud.

La fonction principale est donc la partie de l'éditeur graphique qui reçoit tous les messages de l'environnement MS-Windows et qui connaît tous les schémas existant à un moment donné de l'exécution du programme.

Comme nous l'avons énoncé précédemment, un objet schéma contient une liste d'objets représentant des noeuds et des arcs (le dessin du schéma). Il contient également une liste d'objets représentant les noeuds et arcs (qui sont les ressources du schéma) mis à la disposition de l'utilisateur pour créer les noeuds et arcs du dessin. Autrement dit, un objet schéma maintient

deux listes de noeuds et arcs, une liste représentant le dessin du schéma et une autre représentant les noeuds et arcs qui peuvent être insérés dans le dessin.

Reprenons l'exemple précédent et approfondissons-en les étapes de la création du noeud. Lorsque la fonction principale à reçu le message signalant que l'utilisateur veut créer un noeud, elle demande à l'utilisateur de choisir dans les noeuds des ressources du schéma, le noeud qu'il veut insérer sur le dessin. Ensuite, un noeud du dessin est créé à partir du noeud des ressources. Ce nouveau noeud du dessin reprend toutes les informations graphiques contenues dans le noeud des ressources (liste des éléments graphiques définissant le contour, liste des éléments fixes, ...). Les informations non graphiques et communes à tous les noeuds du dessin provenant de ce même noeud des ressources ne sont pas dupliquées (noms des fonctions du serveur à appeler lors d'un déplacement, d'une suppression, ...). Toutefois, chaque noeud contient une référence vers le noeud des ressources qui est son origine afin de connaître cette information commune. L'information qui n'est pas dupliquée permet d'économiser de la place en mémoire.

Pour la création des arcs sur un dessin, ainsi que pour les éléments fixes des arcs ou de noeuds, nous procédons de la même manière. Autrement dit, chaque arc et élément fixe d'un noeud ou d'un arc du dessin contiennent une référence sur l'arc ou l'élément fixe respectivement des ressources qui est son origine.

La création d'un arc par l'utilisateur est réalisée de la manière suivante. L'utilisateur désigne les noeuds et/ou arcs sur lesquels l'arc à insérer doit porter. L'arc est inséré sur les éléments le supportant. Les points d'ancrage de l'arc sont les deux points les plus proches l'un de l'autre.

L'arc inséré est une copie de l'arc des ressources, redimensionnée pour qu'elle se positionne sur les deux points précédemment cités. Après que l'arc soit inséré, l'utilisateur le repositionne à sa guise pour qu'il se place comme il le désire. Pour illustrer cela, prenons comme exemple l'insertion d'un rôle entre un T.E. et un T.A. d'un schéma E/A. L'utilisateur désigne le T.E. et le T.A. sur lesquels il veut insérer le rôle. Puis, il désigne l'arc représentant un rôle dans les ressources. Le rôle inséré est disposé entre les points d'ancrage du T.E. et du T.A. qui sont les plus proches l'un de l'autre. Ce rôle inséré est illustré à la figure 37.(a). L'utilisateur peut alors modifier l'arc pour qu'il se place comme nous l'illustrons à la figure 37.(b).

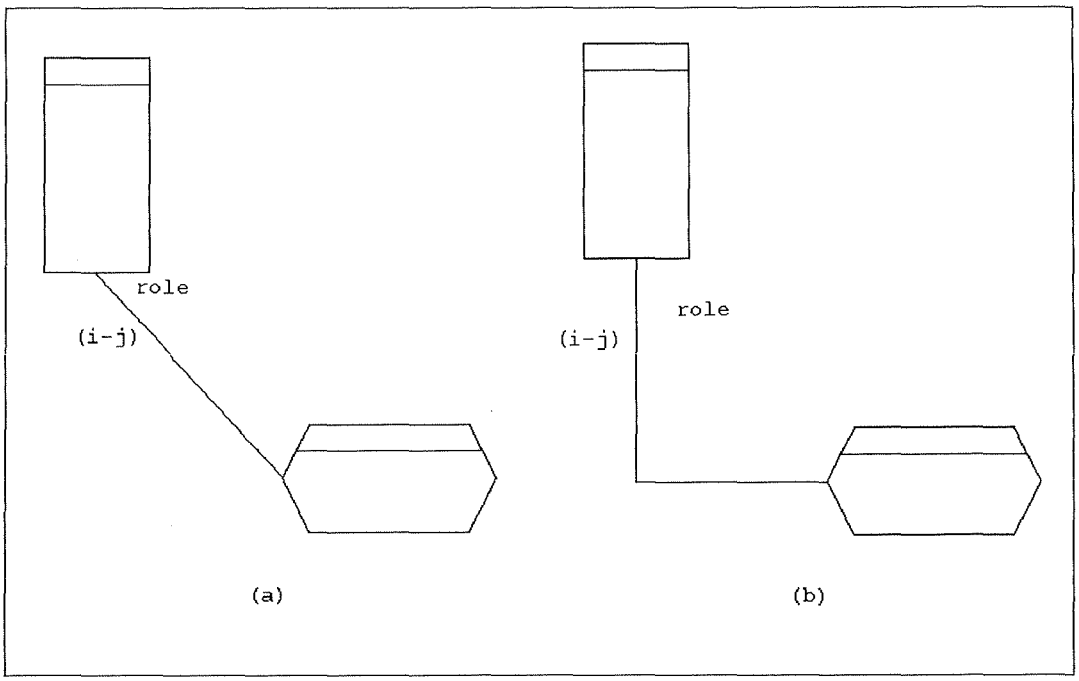

Fig. 37 : Création d'un arc

Pour chaque type de schéma, les noeuds et arcs des ressources sont décrits dans un fichier (ce fichier est noté fichier de ressources). Le fichier de ressources est un fichier ASCII contenant la description des noeuds, arcs et droites complexes qui sont mis à la disposition de l'utilisateur pour la construction des dessins (la description de ce fichier est donnée au point 4.2. intitulé " Fichier de ressources ").

Lors de la création d'un schéma, l'utilisateur désigne le fichier de ressources qu'il veut utiliser pour construire le dessin du schéma. Le schéma lit le fichier de ressources et crée des objets noeuds, arcs et droites complexes dont la définition est donnée dans ce fichier. Lorsque ce fichier est lu, le schéma contient la liste des noeuds, arcs et droites complexes représentant ses ressources.

En ce qui concerne les éléments graphiques statiques et dynamiques, nous procédons de la manière suivante. Tout élément graphique est défini par une classe. Ces classes contiennent toutes les informations nécessaires à la gestion des éléments graphiques qu'elles représentent. Par exemple pour un élément graphique représentant une droite, sa classe contient les deux points extrémités de la droite, son épaisseur, sa couleur, ...

Les objets représentant des éléments graphiques sont donc entièrement redéfinis. Nous n'utilisons MS-Windows que pour le tracé des éléments graphiques sur le dessin. Par exemple,

pour tracer l'élément graphique représentant une droite, nous appelons la fonction LineTo de MS-Windows qui trace un segment de droite entre deux points.

Nous allons exposer à présent le dernier élément intervenant dans l'architecture de l'éditeur graphique, à savoir le serveur.

Les requêtes que l'éditeur graphique soumet au serveur sont réalisées de la manière suivante.

Lors de la création et de la suppression d'un schéma, les méthodes correspondant à l'objet schéma émettent les requêtes d'initialisation et de terminaison, respectivement, de la communication (le type d'information échangée est décrit au point 4.3.3. intitulé " Type d'information échangée ").

Lorsqu'une action est réalisée sur un élément du dessin d'un schéma et que cette action doit être communiquée au serveur (par exemple : insérer un élément, le modifier, ...), nous procédons de la manière suivante. La méthode de l'objet correspondant à l'action envoie un message au serveur en lui fournissant des informations (elles sont décrites au point 4.3.3. intitulé " Type d'information échangée "). Ces informations contiennent entre autre le nom (une chaîne de caractères) de la fonction du serveur qui doit être appelée. Ce nom est contenu dans l'objet des ressources du schéma correspondant à l'objet concerné par l'action.

Après avoir transmis cette requête, l'éditeur graphique attend la réponse du serveur. Dans le cas où la réponse est positive, l'éditeur graphique exécute l'action demandée sur l'objet. Dans le cas contraire, il ne l'exécute pas.

Pour permettre de bien comprendre cette interaction entre l'éditeur graphique et le serveur, nous allons l'illustrer par un exemple. Prenons le cas de la modification d'un noeud du dessin d'un schéma. La fonction principale de l'éditeur graphique appelle la méthode de modification du noeud. Cette méthode demande au noeud des ressources qui est l'origine du noeud, la chaîne de caractères désignant le nom de la fonction de modification du serveur à appeler. A ce moment, il envoie un message au serveur en lui fournissant, entre autre, cette chaîne de caractères. Si la réponse du serveur est affirmative, la méthode modifie les valeurs des chaînes de caractères que le serveur lui a fourni et rend la main à l'application. Si la réponse du serveur est négative, la méthode rend directement la main à l'application sans rien modifier.

Lorsque le serveur émet des requêtes à l'éditeur graphique, celles-ci sont transmises à la fonction principale de ce dernier. La fonction principale exécute l'action correspondant à la requête et rend la main à MS-Windows.

4.2. Fichier de ressources

Pour permettre la généralité du point de vue des types de schémas que le logiciel doit être capable de représenter, EGOO se sert d'un fichier de ressources décrivant les composants (c'est-à-dire les types de noeuds, d'arcs et de droites complexes) qui peuvent être utilisés pour la construction d'un schéma. Des exemples de fichiers de ressources pour les modèles E/A et diagramme des flux sont donnés en annexe.

La suite de cette partie expose les éléments pouvant être définis dans le fichier de ressources.

4.2.1. Conventions utilisées

Pour la description du contenu du fichier de ressources, les conventions suivantes seront utilisées :

NOEUD : ce qui est en majuscule est un mot clé du fichier de ressources.

nom_du_noeud : ce qui est en minuscule est ce qui est donné par l'utilisateur décrivant le fichier de ressources.

{} : ce qui est inscrit entre accolades peut être répété autant de fois que cela s'avère nécessaire.

[] : ce qui est représenté entre crochets fournit un choix possible de valeurs (ces valeurs étant séparées par des virgules).

4.2.2. Structure générale

La structure générale du fichier de ressources a la forme suivante :

```
SCHEMA nom_du_schéma

DROITE nom_de_la_droite
  BEGIN
   .....
  END

NOEUD nom_du_noeud
  BEGIN
   .....
  END

ARC nom_de_l_arc
  BEGIN
   .....
```

```
 END
% commentaire
```

L'élément SCHEMA nom_du_schéma fournit le nom du schéma qui est décrit.

L'élément DROITE permet la définition d'une droite complexe qui pourra être utilisée pour la définition des éléments NOEUD et ARC. Sa définition est donnée au point 4.2.3.

Les éléments NOEUD et ARC définissent respectivement des éléments du type noeud et arc du schéma. Leurs définitions seront données dans les points 4.2.4. et 4.2.5.

Les mots BEGIN et END permettent de définir le corps des éléments.

La possibilité d'introduire des commentaires dans le fichier de ressources est donnée par l'introduction d'un caractère "%" dans une ligne. Tous les caractères introduits après le caractère "%" de la ligne sont ignorés.

Les noms des noeuds, arcs, droites et schémas sont définis sur vingt caractères.

Les éléments DROITE, NOEUD et ARC sont définis par rapport à un point de coordonnées (0,0). L'axe horizontal des X croît vers la droite et l'axe vertical des Y croît vers le bas. L'unité de mesure est le dixième de millimètre.

4.2.3. DROITE (COMPLEXE)

L'élément DROITE permet la définition d'une droite complexe. Cette droite complexe est composée d'un contour (une droite ou une multi-droite) ainsi que d'ensembles d'éléments graphiques statiques (éléments composés). Cette droite peut être utilisée lors de la définition des noeuds et arcs, que ce soit pour le contour ou pour les composants fixes. La structure de la définition d'une droite est la suivante :

```
DROITE nom_de_la_droite
 BEGIN
  nom_element_cont {parametres}
  ANCRAGE = X1 Y1 X2 Y2
  {
   BEGIN
    ROTATE = [YES,NO]
    CONTRAINTE = X Y
    {nom_element {parametres} }
   END
  }
 END
```

a) nom_de_la_droite

La droite est identifiée par son nom, nom_de_la_droite. Ce nom est un identifiant parmi les noms de toutes les droites qui peuvent êtres définies dans le fichier de ressources.

b) nom_element_cont {parametres}

Le nom de l'élément qui définit le contour de la droite est donné par nom_element_cont. Cet élément définissant un segment de droite doit obligatoirement être du type droite ou multi-droite (Cf. 4.2.6). Dans le cas d'une multi-droite, celle-ci ne peut être composée que d'un seul segment lors de sa définition.

c) ANCRAGE = X1 Y1 X2 Y2

ANCRAGE définit les deux points de coordonnées (X1,Y1) et (X2,Y2) qui sont les points extrémités de la droite qui est définie.

d) { nom_element {parametres} }

nom_element définit un élément qui peut être ajouté à l'élément composé de la droite complexe. Cet élément composé est entouré d'un BEGIN et d'un END. Cet ensemble des éléments dans le BEGIN ... END, élément composé, définit un et un seul élément qui ne peut être scindé en ses composants.

CONTRAINTE définit la contrainte du composant. Un composant peut être défini avec une contrainte par rapport à un des deux points d'ancrage de la droite : (X,Y). La distance de ce composant au point (X,Y) sera toujours identique quelle que soit la déformation de la droite et quelle que soit sa position. Lors d'une rotation de la droite, la contrainte est interprétée comme suit : la distance de l'élément au point d'ancrage (X,Y) est respectée dans le sens et la direction de la nouvelle position de la droite. Un composant ne possédant aucune contrainte est un composant dont les coordonnées restent dans les mêmes proportions par rapport au segment de droite.

ROTATE mis à YES permet de spécifier si l'élément composé doit subir lors d'une rotation de la droite cette même rotation. Mis à NO, l'élément ne subit pas de rotation lors de la rotation de la droite

Remarque : Les éléments composant la droite ne sont pas redimensionnés lorsque la droite est redimensionnée (excepté pour l'élément nom_element_cont).

4.2.4. NOEUD

L'élément noeud est un des deux éléments principaux pour la construction d'un schéma. Un noeud est un élément composé d'un contour défini par un ensemble d'éléments graphiques statiques (droite, rectangle, ellipse, ...). Il est aussi composé d'éléments fixes, c'est-à-dire d'éléments qui existent avec le noeud. Ces composants ne peuvent être ni effacés, ni ajoutés au noeud. Ils existent pendant toute la durée de vie du noeud (la définition des éléments fixes a été donnée à la page 26).

Le noeud peut également être composé d'éléments admissibles qui sont des éléments noeuds et arcs définis dans le fichier de ressources. Ces composants admissibles ne sont pas créés lors de la création du noeud, il faut les créer et les associer au noeud pour qu'ils fassent partie de l'ensemble des composants du noeud. Etant donné ce concept de composant admissible, un noeud peut contenir un ensemble d'arcs et de noeuds (la définition de composants admissibles a été donnée à la page 27).

La structure de la définition du noeud est la suivante :

```
NOEUD nom_du_noeud
 BEGIN
  CONTOUR
    TYPE_MOVE_COMP_ADM = [ALL,CONTOUR]
    TYPE_RESIZE_COMP_ADM = [ALL,CONTOUR]
    POSS_MOVE = [UNLIMITED,LIMITED,ALERTE]
    MODIF : FCT = nom_de_la_fonction
    NAVIG : FCT = nom_de_la_fonction
    MOVE : FCT = nom_de_la_fonction
    INFO : FCT = nom_de_la_fonction
    DELETE : FCT = nom_de_la_fonction
    INSERT : FCT = nom_de_la_fonction
    ANCRAGE = X1 Y1 {Xi Yi}
    {nom_element_cont {parametre}}
  COMPOSANTS_FIXES
    {
     Nom
     nom_de_element {parametre}
      TYPE_SENS_RESIZE = [VERT,HORI,VERTHORI, ]
      POSS_MOVE = [YES,NO]
      POSS_MOVE_GLOBAL = [VERT,HORI,VERTHORI, ]
      TYPE_SENS_RESIZE_GLOBAL = [VERT,HORI,VERTHORI, ]
      POSS_MOVE_RESIZE_GLOBAL = [VERT,HORI,VERTHORI, ]
      CONTRAINTE = X Y
      MODIF : FCT = nom_de_la_fonction
    }
  COMPOSANTS_ADMISSIBLES
    { nom_de_element }
END
```

Chaque élément noeud défini dans le fichier de ressources est identifié de manière unique par son nom : nom_du_noeud.

Etudions à présent la structure de la définition du contour, des composants fixes et des composants admissibles d'un noeud.

a) CONTOUR :

Chaque noeud possède un contour constitué d'éléments graphiques statiques. Ces éléments forment une entité unique qui désigne de manière graphique le noeud. Toute action sur cette entité est donc une action réalisée sur le noeud.

Les éléments définis dans le contour ont la signification suivante :

1. TYPE_MOVE_COMP_ADM :

TYPE_MOVE_COMP_ADM peut prendre les valeurs ALL et CONTOUR. Si TYPE_MOVE_COMP_ADM a la valeur ALL, une action de déplacement du contour du noeud entraîne un déplacement de l'ensemble de ses composants admissibles. S'il a la valeur CONTOUR, seul le contour du noeud est déplacé.

2. TYPE_RESIZE_COMP_ADM :

TYPE_RESIZE_COMP_ADM peut prendre les valeurs ALL et CONTOUR. Si TYPE_RESIZE_ADM a la valeur ALL, une action de modification en taille de l'élément porte sur le contour du noeud ainsi que sur l'ensemble de ses composants admissibles. Si par contre il a la valeur CONTOUR, la modification en taille ne porte que sur le contour.

3. POSS_MOVE :

POSS_MOVE peut prendre les valeurs UNLIMITED, LIMITED et ALERTE. Si POSS_MOVE a la valeur UNLIMITED, le déplacement du noeud peut se faire n'importe où sur le schéma. S'il a la valeur LIMITED, le déplacement est limité à l'intérieur du contour du noeud père, et dans ce cas, aucune fonction du serveur n'est appelée. Si par contre il a la valeur ALERTE, il peut être déplacé de l'intérieur du contour du noeud père vers l'intérieur du contour d'un autre noeud, et la fonction définie dans la close MOVE est appelée.

4. NAVIG, MODIF, MOVE, INFO, DELETE, INSERT :

Une action réalisée sur le noeud peut être soit une navigation (NAVIG), une modification (MODIF), un déplacement (MOVE) (Cf. point précédent), une demande d'information (INFO), une suppression (DELETE) ou une insertion du noeud dans le schéma (INSERT). Pour chaque action sur le noeud, nous pouvons définir la fonction associée :

FCT = nom_de_la_fonction : nom_de_la_fonction est le nom de la fonction du serveur qui sera appelée par le programme EGOO lorsqu'une action correspondante a été réalisée sur le noeud. Si le serveur ne doit pas être appelé pour une action, le nom de la fonction de l'action correspondante doit être mis à NULL. Toutefois, la fonction INSERT est obligatoire.

5. ANCRAGE :

Chaque noeud doit posséder un ou plusieurs points d'ancrage ((X1,Y1), (X2,Y2), ...) où peuvent venir se fixer les arcs portant sur le noeud. Le nombre de points d'ancrage est limité à 10.

6. {nom_element_cont {parametre}} :

{nom_element_cont [parametre]} est l'ensemble des éléments graphiques statiques qui définissent le contour du noeud. Cet ensemble d'éléments ne peut contenir d'éléments demandant des informations au serveur. Autrement dit, il ne peut contenir les éléments dynamiques du type texte dynamique et listebox. La définition précise de ces éléments est donnée au point 4.2.6.

b) COMPOSANTS_FIXES :

COMPOSANTS_FIXES définit l'ensemble des éléments fixes qui composent le noeud. Ces éléments sont appelés composants fixes car lors de la création du noeud, ils sont automatiquement créés. Ces éléments ne peuvent être ni ajoutés, ni supprimés.

Pour chaque composant fixe, nous pouvons définir :

1. Nom :

Nom est le nom du composant qui l'identifie parmi tous les noms des composants du noeud.

2. nom_de_element :

nom_de_element est le nom de l'élément graphique qui représente le composant fixe.

3. TYPE_SENS_RESIZE :

TYPE_SENS_RESIZE précise le(s) sens de redimensionnement du composant fixe lorsque celui-ci est sélectionné pour un redimensionnement. Ce paramètre peut prendre les valeurs VERT, HORI, VERTHORI ou aucune des ces trois valeurs. Si TYPE_SENS_RESIZE a la valeur VERT, la taille de l'élément ne peut être modifiée que dans le sens vertical. Mis à HORI, la modification en taille ne peut se faire que dans le sens horizontal. Mis à VERTHORI, la taille de l'élément peut être modifiée dans le sens horizontal et vertical. Si TYPE_SENS_RESIZE ne prend aucune des trois valeurs précédentes, aucune modification de taille de l'élément n'est permise .

4. POSS_MOVE :

POSS_MOVE précise la possibilité ou non de déplacer le composant fixe lorsque celui-ci est sélectionné. Ce paramètre peut prendre les valeurs YES ou NO. Si POSS_MOVE a la valeur YES l'élément fixe peut être déplacé par l'utilisateur. Mis à NO, ce dernier ne peut déplacer l'élément fixe.

5. POSS_MOVE_GLOBAL :

POSS_MOVE_GLOBAL précise la possibilité de déplacer le composant fixe lorsque le noeud est sélectionné et déplacé. Ce paramètre peut prendre les valeurs VERT, HORI, VERTHORI ou aucune de ces trois valeurs. Si POSS_MOVE_GLOBAL a la valeur VERT (respectivement HORI), le composant fixe ne peut suivre le déplacement du contour du noeud que dans le sens vertical (respectivement horizontal). Mis à la valeur VERTHORI, le composant fixe suit le déplacement du contour dans les deux sens. Si le paramètre n'est à aucune de ces trois valeurs, le composant fixe reste sur place tandis que le contour est déplacé.

6. TYPE_SENS_RESIZE_GLOBAL :

TYPE_SENS_RESIZE_GLOBAL précise le(s) sens de redimensionnement du composant fixe lorsque le noeud est sélectionné et redimensionné. Ce paramètre peut prendre les valeurs VERT, HORI, VERTHORI ou aucune de ces trois valeurs. Si TYPE_SENS_RESIZE_GLOBAL a la valeur VERT, la taille de l'élément est modifiée dans le sens vertical lors d'un redimensionnement du noeud. Mis à HORI, la modification en taille n'est réalisée que dans le sens horizontal. Mis à VERTHORI, la taille de l'élément est modifiée dans le sens horizontal et vertical. Si TYPE_SENS_RESIZE_GLOBAL ne prend aucune des trois valeurs précédentes, aucune modification de taille n'est effectuée sur l'élément.

7. POSS_MOVE_RESIZE_GLOBAL :

POSS_MOVE_RESIZE_GLOBAL précise la possibilité de déplacement du composant fixe dû au redimensionnement du noeud lorsque le noeud est sélectionné et redimensionné. Ce paramètre peut prendre les valeurs VERT, HORI, VERTHORI ou aucune de ces trois valeurs. Si ce paramètre a la valeur VERT, l'élément fixe ne peut être déplacé que verticalement lorsque le noeud est redimensionné, mais la distance horizontale par rapport au point de contrainte (défini ci-dessous) est maintenue. Mis à HORI, le déplacement ne peut se faire que dans le sens horizontal, mais la distance verticale par rapport au point de contrainte est maintenue. Mis à VERTHORI, le déplacement de l'élément peut être effectué dans le sens horizontal et vertical. Si POSS_MOVE_RESIZE_GLOBAL ne prend aucune des trois valeurs précédentes, la distance horizontale et verticale par rapport au point de contrainte est inchangée après redimensionnement.

8. CONTRAINTE :

CONTRAINTE définit le point (X , Y) par rapport auquel le composant fixe ne peut être déplacé horizontalement, verticalement ou encore horizontalement et verticalement si la variable POSS_MOVE_RESIZE_GLOBAL vaut VERT ou HORI ou rien respectivement.

9. MODIF :

MODIF est tel que si le composant fixe est modifié, la fonction de la close MODIF du serveur est appelée. Si la fonction est mise à NULL, le serveur n'est pas appelé.

c) COMPOSANTS_ADMISSIBLES :

Les composants admissibles du noeud sont les éléments du type noeud ou arc qui peuvent être ajoutés ou supprimés au noeud. nom_de_element est le nom du noeud ou de l'arc défini dans le fichier de ressources.

4.2.5. ARC

L'arc est le second élément principal pour la construction d'un schéma. C'est un élément du schéma qui permet de relier deux noeuds ou deux autres arcs. L'arc est composé obligatoirement d'un élément graphique statique qui définit son contour (droite, multi-droite, râteau). D'autres éléments, les éléments fixes, que ce soient des éléments graphiques statiques ou dynamiques, peuvent venir se disposer sur et autour du contour. Les extrémités du contour définissent les points où l'arc se fixe sur les noeuds sur lesquels il porte.

Un arc est défini dans le sens horizontal. Une rotation de l'arc effectuée par un utilisateur dans le programme EGOO entraîne une rotation de tous les éléments qui composent l'arc.

La structure de la définition de l'arc est la suivante :

```
ARC nom_de_arc
 BEGIN
  CONTOUR
   MODIF : FCT = nom_de_la_fonction
   INFO : FCT = nom_de_la_fonction
   DELETE : FCT = nom_de_la_fonction
   INSERT : FCT = nom_de_la_fonction
   nom_element_cont {parametre}
  COMPOSANTS_FIXES
   {
    Nom
    nom_de_element {parametre}
     TYPE_SENS_RESIZE = [VERT,HORI,VERTHORI, ]
     POSS_MOVE = [YES,NO]
     POSS_MOVE_GLOBAL = [VERT,HORI,VERTHORI, ]
     TYPE_SENS_RESIZE_GLOBAL = [VERT,HORI,VERTHORI, ]
     POSS_MOVE_RESIZE_GLOBAL = [VERT,HORI,VERTHORI, ]
     CONTRAINTE = X Y
     MODIF : FCT = nom_de_la_fonction
   }
  NOEUD_ARC_ADMISSIBLES
   { nom_de_element }
 END
```

L'arc est défini par un nom, nom_de_arc, qui l'identifie de manière unique parmi tous les noms des arcs du fichier de ressources.

a) CONTOUR :

Le contour est le squelette de l'arc. Il est constitué d'un élément graphique statique. Toute action sur cet élément graphique est une action réalisée sur l'arc.

Les éléments définis dans le contour ont la signification suivante :

Les actions MODIF, INFO, DELETE et INSERT sont identiques à celles définies pour le noeud.

nom_element_cont est le nom de l'élément graphique statique définissant le contour. La définition précise de cet élément est donnée au point 4.2.6.

b) COMPOSANTS_FIXES :

COMPOSANTS_FIXES définit l'ensemble des éléments qui composent l'arc de manière statique. La déclaration est identique à celle des composants fixes du noeud.

c) NOEUD_ARC_ADMISSIBLES :

NOEUD_ARC_ADMISSIBLES précise l'ensemble des noeuds et arcs définis dans le fichier de ressources et sur lesquels peut porter l'arc en cours de définition. nom_de_element désigne le nom d'un élément de cet ensemble.

4.2.6. Eléments graphiques

Les éléments graphiques sont des éléments prédéfinis (parmi les éléments droite, multi-droite, rectangle, ellipse, polygone, texte, listebox, ...) qui peuvent êtres utilisés lors de la définition des noeuds et des arcs.

La définition de ces éléments graphiques est donnée dans les points suivants :

a) Droite :

L'élément graphique statique droite est en fait un segment de droite. La droite répond à la définition suivante :

DROITE X1 Y1 X2 Y2 style_pinceau larg_pinceau couleur_pinceau style_droite

où

- X1, Y1, X2 et Y2 sont les coordonnées des extrémités (X1,Y1) et (X2,Y2) du segment de droite.

- style_pinceau est le style du pinceau utilisé pour tracer la droite. Il peut prendre les valeurs suivantes (la figure 38 illustre le type de style_pinceau) :

  - SOLID : droite continue
  - DASH : droite discontinue
  - DOT : droite représentée par une succession de points
  - DASHDOT : alternance d'un segment de droite et d'un point
  - DASHDOTDOT : alternance d'un segment de droite et de deux points
  - NULL : droite invisible.

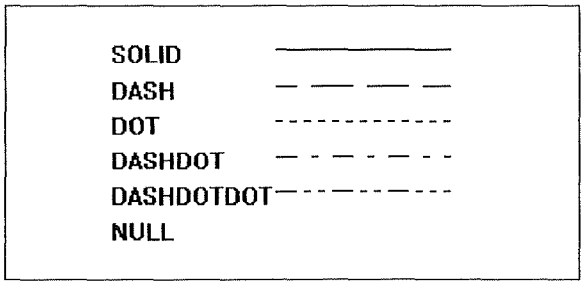

Fig. 38 : Styles de pinceau

- larg_pinceau est la largeur du pinceau.

- couleur_pinceau est la couleur du pinceau pour le tracé de la droite. Cette couleur est spécifiée avec les trois couleurs de base ROUGE,VERT,BLEU. Chaque couleur de base est donnée par un nombre compris entre 0 et 255. 0 signifie que la couleur de base n'est pas utilisée. Par contre, 255 signifie que la couleur de base est utilisée dans sa teinte la plus prononcée. Tout nombre compris entre 0 et 255 exprime donc la variation de l'intensité de la couleur de base. Notons que 0 0 0 donne la couleur noire et 255 255 255 donne la couleur blanche.

- style_droite est le style de la droite. Il peut prendre les valeur suivantes :

  - SIMPLE : la droite est une droite simple (autrement dit sans flèche)

  - FLECHEG : la droite possède une flèche à sa gauche

  - FLECHED : la droite possède une flèche à sa droite

  - FLECHEGD : la droite possède une flèche à gauche et à droite.

b) Multi-droite :

Une multi-droite est un élément graphique statique composé d'une suite de segments de droites mis bout à bout. La multi-droite répond à la définition suivante :

MULTIDROITE NBRPTS X1 Y1 X2 Y2 [Xi Yi] style_pinceau larg_pinceau couleur_pinceau style_droite

où

- NBRPTS est le nombre de points de la droite.

- X1, Y1, X2, Y2, ... Xi, Yi, ... sont les points extrémités des segments de la multi-droite.

- les autres paramètres sont identiques à ceux définis pour la droite.

c) Rectangle :

Un rectangle est un élément graphique statique représentant un rectangle. La restriction suivante est donnée : les côtés du rectangle sont toujours parallèles aux axes du système de coordonnées. La définition du rectangle est :

RECTANGLE X1 Y1 X2 Y2 style_pinceau larg_pinceau couleur_pinceau style_brosse couleur_brosse hatch_brosse

où

- X1, Y1, X2 et Y2 sont les coordonnées des coins supérieur gauche (X1,Y1) et inférieur droit (X2,Y2) du rectangle.

- style_pinceau, larg_pinceau et couleur_pinceau ont des significations identiques à celles définies pour la droite.

- style_brosse est le style de la brosse pour le tracé de l'intérieur du rectangle. Il peut prendre les valeurs suivantes :

  - SOLID : brosse solide
  - HOLLOW : brosse invisible
  - HATCHED : brosse en hachurage.

- couleur_brosse est la couleur de la brosse définie de façon identique à la couleur du pinceau.

- hatch_brosse est le style de hachurage dont les valeurs sont les suivantes (la figure 39 illustre le type de hatch_brosse) :

  - HORIZONTAL : hachurage horizontal
  - VERTICAL : hachurage vertical
  - FDIAGONAL : hachurage en diagonal incliné à -45°
  - BDIAGONAL : hachurage en diagonal incliné à 45°
  - CROSS : hachurage vertical et horizontal
  - DIAGCROSS : hachurage en diagonal incliné à -45° et à 45°.

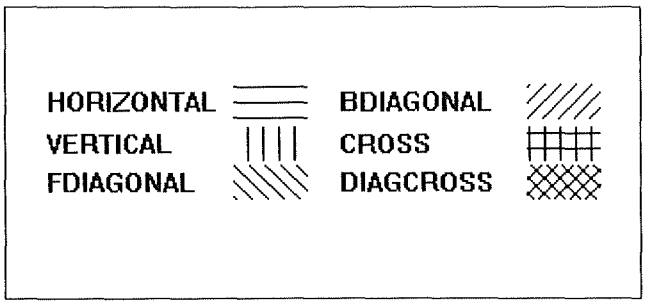

Fig. 39 : Styles d'hachurage

d) Ellipse :

Une ellipse est un élément graphique statique représentant une ellipse. Sa définition est la suivante :

ELLIPSE X1 Y1 X2 Y2 style_pinceau larg_pinceau couleur_pinceau style_brosse hatch_brosse couleur_brosse

où

- les paramètres sont identiques à ceux définis pour le rectangle.

- les points (X1,Y1) et (X2,Y2) sont les coordonnées du plus petit rectangle contenant l'ellipse.

e) Polygone :

Un polygone est un élément graphique statique représentant un polygone. Sa définition est la suivante :

POLYGONE NBRPTS X1 Y1 X2 Y2 [Xi Yi] style_pinceau larg_pinceau couleur_pinceau style_remplissage style_brosse couleur_brosse hatch_brosse

où

- les paramètres sont identiques à ceux de la droite multiple. De plus, les paramètres style_brosse, hatch_brosse et couleur_brosse définissent comme pour le rectangle les paramètres de la brosse peignant l'intérieur du polygone.

- style_remplissage définit le type de remplissage du polygone et peut prendre les valeurs ALTERNATE ou WINDING. Si style_remplissage a la valeur WINDING, cela signifie que le polygone est rempli dans toutes ses parties intérieures. Cette situation est illustrée à la figure 40.(b). Si par contre style_remplissage est mis à ALTERNATE, le polygone est rempli de façon différente : les parties du polygone qui sont remplies sont celles qui sont accessibles à partir de

l'extérieur en suivant une droite de sorte que cette droite coupent un nombre impair de droites définissant le polygone. Cette situation est illustrée à la figure 40.(a). Pour plus de précisions veuillez vous reporter à l'ouvrage [9].

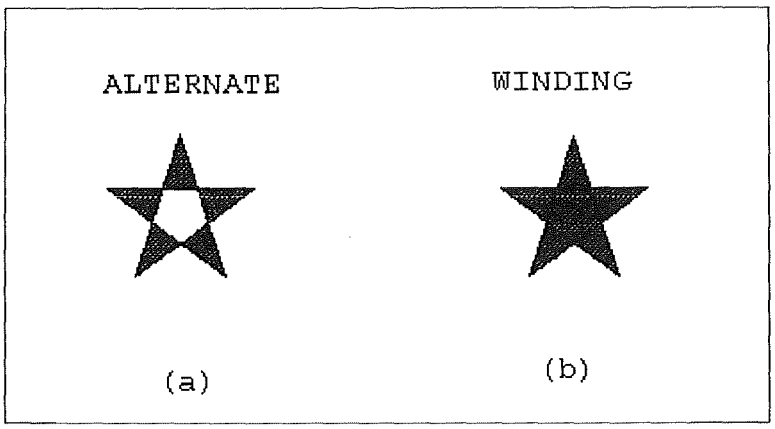

Fig. 40 : Style de remplissage

f) Râteau Vertical :

Le râteau vertical est un élément graphique statique composé d'un ensemble de segments de droites qui peuvent connecter plusieurs points à plusieurs autres points dans le sens vertical. Sa définition est la suivante :

RATEAUV style_pinceau larg_pinceau couleur_pinceau style_droite

où les paramètres style_pinceau, larg_pinceau, couleur_pinceau, style_droite sont ceux définis pour les droites.

Remarquons qu'aucun paramètre relatif au nombre de segments de droites, n'est demandé car nous pouvons connecter un nombre quelconque de points à un autre nombre quelconque de points. Et c'est au moment du positionnement du râteau sur le schéma que l'utilisateur spécifie les noeuds ou arcs sur lesquels repose le râteau et le point milieu du râteau.

g) Râteau horizontal :

Le râteau horizontal est un élément graphique statique composé d'un ensemble de segments de droites qui peuvent connecter plusieurs points à plusieurs autres points dans le sens horizontal. Sa définition est la suivante :

RATEAUH style_pinceau larg_pinceau couleur_pinceau style_droite

où les paramètres style_pinceau, larg_pinceau, couleur_pinceau, style_droite sont ceux définis pour les droites.

h) Texte :

Le texte est un élément graphique définissant une chaîne de caractères. Sa définition est la suivante :

TEXTE X1 Y1 X2 Y2 type hauteur style couleur_pinceau mode couleur-background valeur

où

- X1, Y1, X2, Y2 sont les coordonnées des coins supérieur gauche (X1, Y1) et inférieur droit (X2,Y2) du rectangle contenant la chaîne de caractères.

- type définit si le texte est mis statiquement dans le fichier ou s'il doit être demandé au serveur. type peut prendre les valeurs STATIC ou VARIABLE. Mis à VARIABLE, le texte est demandé au serveur. Mis à STATIC, le paramètre valeur doit être défini et donne la valeur que le texte prendra. Dans le cas où type est mis à STATIC, le texte est un élément graphique statique. Par contre, si type est mis à VARIABLE, le texte est un élément graphique dynamique.

- hauteur définit la hauteur des caractères du texte. L'unité de mesure est le dixième de millimètre.

- style est le style de texte. Il peut prendre les valeurs suivantes :

  - LEFT : le texte est aligné à gauche dans le rectangle.
  - CENTER : le texte est centré dans le rectangle.
  - RIGHT : le texte est aligné à droite dans le rectangle.

- mode définit le mode d'écriture du texte et peut prendre les valeurs OPAQUE ou TRANSPARENT. Mis à OPAQUE, les zones entre les caractères du texte sont remplies par la couleur du fond. Mis à TRANSPARENT, les zones entre les caractères du texte sont transparentes.

- couleur_background définit la couleur du fond du texte.

- valeur est la valeur que prend le texte lorsque le paramètre type est mis à STATIC.

i) Listebox :

Une listebox est un élément graphique dynamique définissant une liste de chaînes de caractères. Sa définition est la suivante :

LISTEBOX X1 Y1 X2 Y2 hauteur style_texte couleur_pinceau mode couleur_background

où

- X1, Y1, X2, Y2 sont les coordonnées des coins supérieur gauche (X1, Y1) et inférieur droit (X2,Y2) du rectangle contenant la listebox.

- hauteur définit la hauteur des caractères du texte. L'unité de mesure est le dixième de millimètre.

- style_texte, couleur_pinceau, mode et couleur_background sont identiques aux paramètres de Texte.

4.3. Implantation de la communication EGOO - Serveur

Cette partie décrit l'implantation de la communication entre le logiciel EGOO et le serveur. Pour rappel, la conception de cette communication a été décrite au point 3.2. intitulé " Communication EGOO - Serveur ". Nous décrirons tout d'abord le principe de " Dynamique Data Exchange " (D.D.E.) que l'environnement MS-Windows offre pour que deux logiciels puissent communiquer. Ensuite, nous décrirons l'utilisation du D.D.E. pour que l'éditeur graphique puisse communiquer avec le serveur. Nous terminerons par l'exposé du type d'information qui est échangée entre ces deux derniers logiciels.

4.3.1. D.D.E.

Avant de commencer l'exposé du principe du D.D.E., il faut savoir que l'environnement MS-Windows possède une architecture basée sur l'envoi de messages. Autrement dit, les interactions entre un programme et l'environnement ou entre deux programmes se font par l'envoi de messages.

Le D.D.E. est basé sur le système de messages de MS-Windows. Deux programmes participent à une conversation D.D.E. lorsque l'un des deux dépose un message à l'intention de l'autre. Ces deux programmes sont désignés par serveur et client. Un serveur D.D.E. est un programme qui dispose de données. Un client D.D.E. est un programme qui demande au serveur des données.

Une conversation D.D.E. est initialisée par le programme client. Le client émet un message d'initialisation à tous les programmes qui sont en train de s'exécuter. Ce message indique une catégorie générale de données dont le client a besoin. Un serveur D.D.E. qui possède cette catégorie de données répond à cette émission de message. A ce point, la conversation commence.

Un seul programme peut être à la fois client d'un programme et serveur d'un autre programme mais cela requiert deux conversations D.D.E. différentes. Un serveur peut délivrer des données à plusieurs clients, et un client peut obtenir des données de plusieurs serveurs. Toutefois, cela demande encore des conversations D.D.E. distinctes.

Il y a trois types de conversation D.D.E. : liaison froide (" cold link "), liaison bouillante (" hot link ") et liaison chaude. (" warm link ").

- Liaison froide :

  Après avoir initialisé la conversation, le client peut demander au serveur des données. Ce dernier ne peut que répondre aux demandes du client. Il n'a pas d'initiative dans cette conversation. Lorsque le client veut terminer la conversation, il signale au serveur la clôture de la conversation.

- Liaison bouillante :

  Le problème avec la liaison froide est le suivant. Lorsque les données du serveur changent de valeur, il ne peut en avertir le client. La conversation de type liaison bouillante va essayer de remédier à ce problème de la façon suivante :

  Après avoir initialisé la conversation, le client signale au serveur quelles sont les données à propos desquelles le client veut être averti de leurs modifications. Lorsqu'une de ces données est modifiée, le serveur en avertit le client. Si le client ne désire plus être averti lors d'une modification de données, il lui suffit de le signaler au serveur. De même, lorsque le client veut mettre fin à la conversation, il n'a qu'à le signaler au serveur.

- Liaison chaude :

  Ce type de conversation combine les conversations liaison froide et liaison bouillante. Autrement dit, le client peut demander des données au serveur et ce dernier peut avertir le client que des données ont été modifiées.

Dans ces trois types de conversations, lorsque le serveur ou le client émet une demande à l'autre programme de la communication, il peut recevoir d'autres messages de l'environnement ou d'autres programmes, ainsi qu'exécuter d'autres opérations avant de recevoir la réponse à sa demande. Autrement dit, lors d'une émission d'une demande de données, le programme peut poursuivre son exécution et traiter d'autres messages avant de recevoir la réponse qu'il attend.

Pour plus de précisions sur le D.D.E. veuillez consulter l'ouvrage [9].

4.3.2. Utilisation du D.D.E. par le logiciel.

La conversation entre l'éditeur graphique et son serveur doit être réalisée de la manière suivante. Après avoir initialisé la conversation, l'éditeur graphique peut demander des services au serveur. Lors d'une demande de service, l'éditeur graphique ne peut rien faire d'autre avant de recevoir la réponse. Le serveur peut demander des services à l'éditeur graphique. Le client est responsable de la terminaison de la communication.

Le type de communication basé sur le principe du D.D.E. ne satisfait pas les exigences demandées par le type de communication entre l'éditeur graphique et son serveur. En effet, la communication D.D.E. permet à une application client ou serveur de poursuivre son exécution lorsqu'elle a envoyé une requête à son serveur ou client respectivement, ce qui est contradictoire avec le fait que l'éditeur graphique et son serveur ne peuvent procéder à d'autres actions avant d'avoir reçu la réponse à sa demande.

Nous allons établir la communication entre l'éditeur graphique et son serveur ainsi que procéder aux différentes requêtes des deux applications de la manière suivante.

- Etablissement de la communication :

  Cette communication est établie en utilisant le principe du D.D.E.. Autrement dit, l'éditeur graphique (client) utilise le D.D.E. pour établir la communication avec son serveur. Cela permet au serveur et à l'éditeur graphique d'obtenir des informations sur l'autre sujet de la communication. Ces informations seront utilisées par la suite pour que client et serveur puissent s'envoyer des requêtes.

- Requête d'une application à l'autre :

  Pour qu'une application puisse émettre une requête à une autre application et rester dans un état bloqué jusqu'à ce qu'elle obtienne la réponse, elle utilise une fonction de MS-Windows qui permet d'envoyer un message et d'attendre la réponse. Ainsi, grâce à cette fonction et aux informations obtenues lors de l'initialisation de la communication, l'éditeur graphique peut demander une requête à son serveur et attendre la réponse avant de poursuivre son exécution. Ce principe est également utilisé pour les requêtes dans le sens serveur vers éditeur graphique.

4.3.3. Type d'information échangée

Pour terminer cette partie relative à l'implantation de la communication entre l'éditeur graphique et son serveur, nous allons préciser le type d'information qui est transmise pour chaque requête.

Nous allons tout d'abord décrire le type d'information échangée lors des requêtes émises par l'éditeur graphique pour le serveur. Pour chacune de ces requêtes, nous citerons d'abord le type d'information fournie au serveur par l'éditeur graphique et ensuite le type d'information renvoyée par le serveur vers l'éditeur graphique.

Initialisation :

- nom du schéma donné dans le fichier de ressources
- nom du fichier de sauvegarde du schéma de l'éditeur graphique.

Réponse

- résultat de la requête (réussite ou échec).

Insérer :

- nom de la fonction du serveur qui est appelée (ce nom est donné dans le fichier de ressources)
- nom du noeud ou de l'arc à insérer (ce nom est le nom du noeud ou de l'arc du fichier de ressources)
- identifiant du père de l'élément concerné par l'insertion. Cet identifiant est fourni si l'utilisateur veut insérer un noeud ou un arc dans un autre noeud.
- identifiants des noeuds ou arcs sur lesquels l'arc doit porter dans le cas où c'est un arc qui doit être inséré.

Réponse :

- résultat de la requête (réussite ou échec).

Si la requête est réussie :

- identifiant de l'élément inséré
- tableau des valeurs des éléments fixes, contenant un élément graphique dynamique, du noeud ou de l'arc inséré.

Le tableau a la forme suivante :

nom_de_élément	valeur
nom_de_élément	valeur
.....	.....
nom_de_élément	valeur

où

- nom_de_élément est le nom de l'élément fixe qui est un élément graphique dynamique
- valeur est la chaîne de caractères de cet élément fixe.

Remarquons que dans le cas d'un élément graphique dynamique représentant une listebox, nom_de_élément est répété autant de fois que la listebox contient de chaînes de caractères.

- nom du contour de l'arc dans le cas où l'élément inséré est un arc.

Supprimer :

- nom de la fonction du serveur qui est appelée
- identifiant du noeud ou arc concerné par l'action.

Réponse :

- résultat de la requête (réussite ou échec).

Modifier :

Si la modification porte sur un noeud ou un arc :

- nom de la fonction du serveur qui est appelée
- identifiant du noeud ou arc concerné par l'action.

Si la modification porte sur un élément fixe :

- nom de la fonction du serveur qui est appelée
- identifiant du noeud ou de l'arc père de l'élément fixe
- nom de l'élément fixe concerné par l'action.

Réponse :

- résultat de la requête (réussite ou échec).

Si la requête est réussie,

- tableau des valeurs des éléments fixes contenant un élément graphique dynamique (dans le cas d'une modification d'un élément fixe, le tableau ne contient que les valeurs de cet élément fixe).

Informer :

- nom de la fonction du serveur qui est appelée
- identifiant du noeud ou arc concerné par l'action.

Réponse :

- résultat de la requête (réussite ou échec).

Si la requête est réussie,

- tableau des valeurs des éléments fixes contenant un élément graphique dynamique du noeud ou de l'arc concerné par l'action.

Naviguer :

- nom de la fonction du serveur qui est appelée
- identifiant du noeud ou arc concerné par l'action.

Réponse :

- résultat de la requête (réussite ou échec).

Si la requête est réussie,

- nom du fichier contenant le schéma que l'éditeur graphique doit afficher.

Déplacer :

- nom de la fonction du serveur qui est appelée
- identifiant du noeud concerné par l'action
- identifiant de l'ancien noeud père du noeud déplacé
- identifiant du nouveau noeud père du noeud déplacé.

Réponse :

- résultat de la requête (réussite ou échec).

Dupliquer

- nom de la fonction du serveur qui est appelée.

Dans le cas d'un noeud,

- identifiant du noeud concerné par l'action.

Dans le cas d'un arc,

- identifiant de l'arc concerné par l'action
- identifiants des nouveaux noeuds ou arcs sur lesquels l'arc dupliqué portera.

Réponse :

- résultat de la requête (réussite ou échec).

Si la requête est réussie,

- identifiant du noeud ou de l'arc dupliqué.

Terminer :

- aucune donnée.

Nous allons à présent décrire le type d'information échangée entre le serveur et l'éditeur graphique lorsque c'est le serveur qui soumet des requêtes à l'éditeur graphique. A nouveau, pour chacune de ces requêtes, nous citerons d'abord le type d'information fournie à l'éditeur graphique par le serveur et ensuite le type d'information renvoyée par l'éditeur graphique vers le serveur.

Insérer :

- identifiant du noeud ou de l'arc à insérer
- identifiant du noeud père de l'élément à insérer si cet élément est un composant admissible d'un noeud

- identifiants des noeuds ou arcs sur lesquels l'arc à insérer porte dans le cas où c'est un arc qui doit être inséré

- tableau des valeurs des éléments fixes contenant un élément graphique dynamique, du noeud ou de l'arc à insérer.

Réponse :

- résultat de la requête (réussite ou échec).

Supprimer :

- identifiant du noeud ou de l'arc à supprimer.

Réponse :

- résultat de la requête (réussite ou échec).

Modifier :

- identifiant du noeud ou de l'arc à modifier

- tableau des valeurs des éléments fixes contenant un élément graphique dynamique (dans le cas d'une modification d'un élément fixe, le tableau ne contient que les valeurs de cet élément fixe).

Réponse :

- résultat de la requête (réussite ou échec).

Déplacer :

- identifiant du noeud ou de l'arc à déplacer

- identifiant de l'ancien noeud père de l'élément à déplacer

- identifiant du nouveau noeud père de l'élément à déplacer.

Réponse :

- résultat de la requête (réussite ou échec).

4.4. Multi-fenêtrage

Cette partie décrit l'implantation du multi-fenêtrage du logiciel EGOO. Ce multi-fenêtrage est réalisé grâce au M.D.I. (" Multiple Document Interface ") de MS-Windows.

Le M.D.I. est une spécification pour les applications qui manipulent des documents dans MS-Windows. La spécification décrit une structure de fenêtres et une interface utilisateur qui permet à l'utilisateur de travailler avec des documents multiples dans une seule et même application. De la même manière que MS-Windows maintient plusieurs fenêtres d'applications sur un seul écran, une application M.D.I. maintient plusieurs documents dans la fenêtre d'une application (Pour plus de précisions sur le principe du M.D.I., veuillez vous reportez à l'ouvrage [9]).

Le M.D.I. permet donc de manipuler dans une même fenêtre (l'application) plusieurs fenêtres filles qui peuvent être manipulées individuellement (déplacées, redimensionnées, mises en icône, ...). Il est à noter que le M.D.I. ne gère pas le tracé d'éléments graphiques dans plusieurs fenêtres identiques. Autrement dit, il n'est pas possible de définir une série de fenêtres qui auraient la particularité de tracer un élément graphique dans toute les fenêtres de cette série lorsque l'élément graphique est tracé dans une seule de celle-ci.

Par conséquent, le M.D.I. est utilisé par le logiciel pour pouvoir manipuler plusieurs fenêtres dans l'application EGOO. Mais, la gestion de plusieurs fenêtres représentant un même schéma doit être réalisée par le logiciel lui-même. Autrement dit, lorsqu'un élément graphique est tracé dans une fenêtre représentant un schéma, le logiciel doit le tracer dans toutes les autres fenêtres représentant ce schéma.

4.5. Classes d'objets

Nous allons décrire dans cette partie les principales classes d'objets composant l'architecture du logiciel EGOO.

Les relations d'héritage des classes sont illustrées à la figure 41.

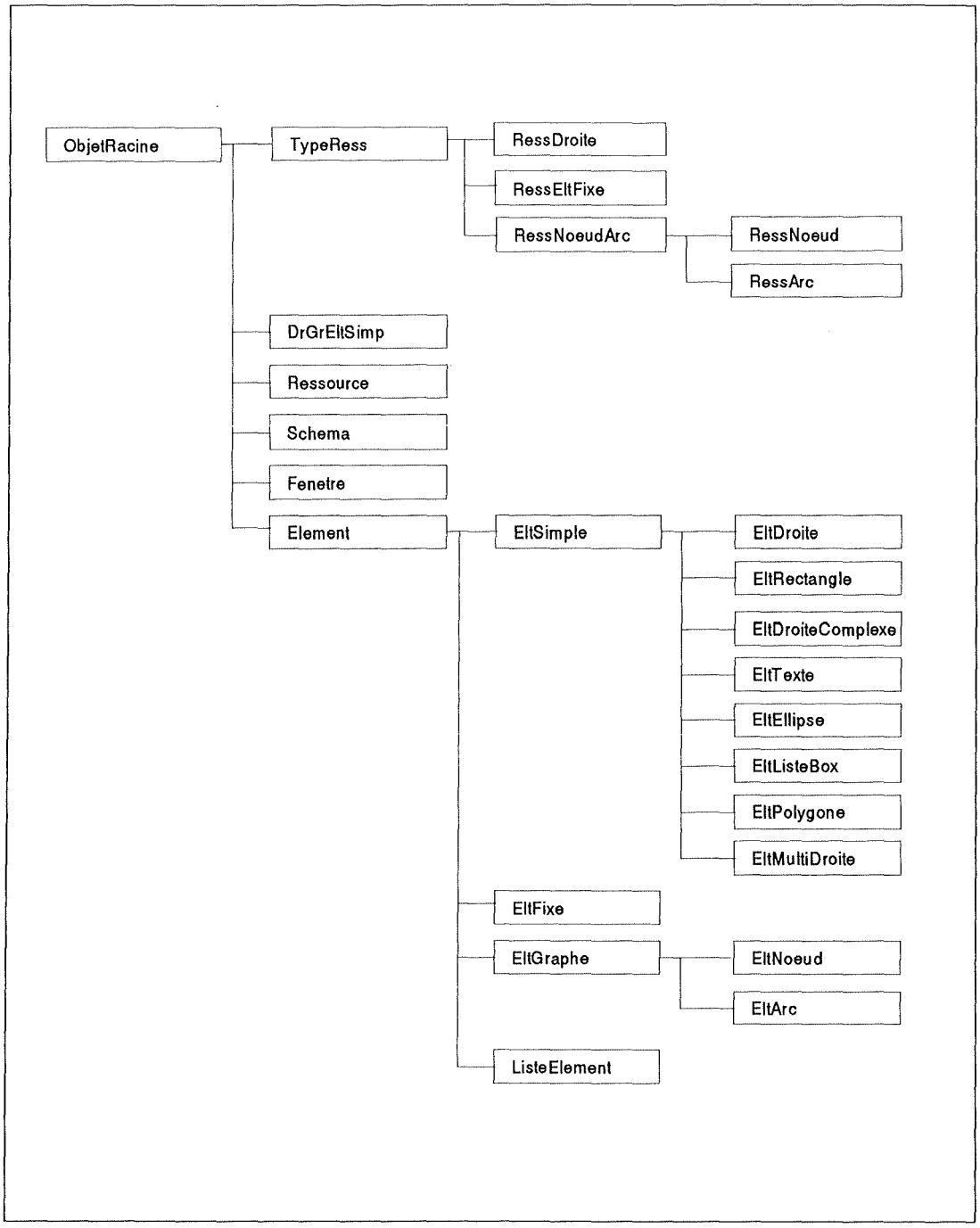

Fig. 41 : Héritage de classes

ObjetRacine :

Classe de base pour toutes les classes de l'architecture du logiciel. Cette classe possède une donnée qui fournit le type de classe dérivée. Cela permet de connaître la classe à laquelle appartient n'importe quel objet. Autrement dit, lorsque nous possédons une référence sur un objet nous pouvons connaître la classe auquel appartient l'objet en question en demandant la donnée fournissant le type de classe dérivée.

Cette classe ne possède que des méthodes d'initialisation et d'accès à ses données.

TypeRess :

Classe de base pour toutes les classes décrivant les éléments du fichier de ressources. Ces éléments sont la droite complexe, l'élément fixe, le noeud et l'arc. Cette classe a pour donnée une chaîne de caractères désignant le nom de l'élément (droite complexe, élément fixe, noeud ou arc) défini dans le fichier de ressources.

Cette classe ne possède que des méthodes d'initialisation et d'accès à ses données.

RessDroite :

Classe définissant une droite complexe du fichier de ressources. La classe contient un objet de classe EltSimple désignant le contour de la droite complexe. Elle contient également une liste d'objets de classe DrGrEltSimple représentant la liste des éléments composés de la droite complexe. Elle contient enfin les deux points d'ancrage de la droite complexe.

Cette classe ne possède que des méthodes d'initialisation et d'accès à ses données.

DrGrEltSimple :

Classe définissant un élément composé d'une droite complexe. Elle possède une liste d'objets de classe EltSimple ainsi que des contraintes de rotation et de distance (par rapport à un des deux points d'ancrage de la droite complexe) de l'élément composé.

Cette classe ne possède que des méthodes d'initialisation et d'accès à ses données.

RessEltFixe :

Classe définissant un élément fixe d'un noeud ou d'un arc. Cette classe contient l'objet de classe EltSimple déterminant l'élément fixe. Elle contient également

- le type de sens de redimensionnement de l'élément fixe,
- la possibilité de déplacement de l'élément fixe,
- la possibilité de déplacement de l'élément fixe lors d'un déplacement du noeud ou de l'arc auquel il appartient,
- la possibilité de déplacement de l'élément fixe lors d'un redimensionnement du noeud ou de l'arc auquel il appartient,

- le type de sens de redimensionnement de l'élément fixe lors d'un redimensionnement du noeud ou de l'arc auquel il appartient,

- le nom de la fonction du serveur à appeler lors d'une modification de l'élément fixe.

Cette classe ne possède que des méthodes d'initialisation et d'accès à ses données.

RessNoeudArc :

Classe de base pour les classes RessNoeud et RessArc. Cette classe contient toutes les données communes aux types de noeuds et d'arcs définis dans le fichier de ressources, à savoir :

- le nom de la fonction du serveur à appeler lors d'une modification du noeud ou de l'arc,

- le nom de la fonction du serveur à appeler lors d'une suppression du noeud ou de l'arc,

- le nom de la fonction du serveur à appeler lors d'une insertion du noeud ou de l'arc,

- le nom de la fonction du serveur à appeler lors d'une demande d'information à propos du noeud ou de l'arc,

- la liste d'objets de classe EltFixe désignant la liste des éléments fixes du noeud ou de l'arc.

Cette classe ne possède que des méthodes d'initialisation et d'accès à ses données.

RessNoeud :

Classe définissant un noeud du fichier de ressources. Les données de cette classe sont les suivantes :

- le type de déplacement des composants admissibles du noeud lors du déplacement de celui-ci,

- le type de redimensionnement des composants admissibles du noeud lors du redimensionnement de celui-ci,

- la possibilité de déplacement du noeud de son noeud père vers un autre noeud,

- le nom de la fonction du serveur à appeler lors d'une navigation sur le noeud,

- le nom de la fonction du serveur à appeler lors d'un déplacement du noeud de son noeud père vers un autre noeud,

- la liste des points d'ancrage du noeud,
- la liste des composants admissibles du noeud,
- la liste des objets de classe EltSimple définissant le contour du noeud.

Cette classe ne possède que des méthodes d'initialisation et d'accès à ses données.

RessArc :

Classe définissant un arc du fichier de ressources. Cette classe a pour données un objet de classe EltSimple définissant le contour de l'arc.

Cette classe ne possède que des méthodes d'initialisation et d'accès à ses données.

Ressource :

Classe contenant toutes les définitions de noeud, arc et droite complexe d'un fichier de ressources. Elle contient donc

- la liste des droites complexes (liste d'objets de classe RessDroite),
- la liste des noeuds (liste d'objets de classe RessNoeud),
- la liste des arcs (liste d'objets de classe RessArc).

De plus, elle contient comme données le nom du fichier de ressources ainsi que le nom du schéma donné dans ce fichier de ressources.

Cette classe ne possède que des méthodes d'initialisation et d'accès à ses données.

Fenetre :

Classe représentant une fenêtre de MS-Windows pour le logiciel EGOO. Elle ne redéfinit pas une fenêtre de MS-Windows, mais elle contient les données nécessaires pour que toutes les fonctions graphiques puissent "dessiner" dans la fenêtre qu'elle représente.

Schéma :

Classe représentant un schéma. Elle contient comme données :

- un objet de classe Ressource donnant les ressources qui peuvent être utilisées pour élaborer le dessin du schéma,

- un objet de classe ListeElement contenant une liste de noeuds et d'arcs. Cette liste de noeuds et arcs représente le dessin du schéma,

- une liste d'objets de classe Fenetre représentant la liste des fenêtres affichant le dessin du schéma,

- le nom du fichier où est enregistré le schéma.

Cette classe possède des méthodes d'initialisation, de modification et de consultation de ses données.

Element :

Classe de base de toutes les classes définissant des concepts du logiciel EGOO (noeud, arc, élément fixe, droite, rectangle, ...). Elle contient pour données :

- une référence sur l'objet schéma auquel appartient l'objet en question,

- une référence sur l'objet père de l'objet en question. Cet objet père est l'objet qui possède comme donnée l'objet en question,

- la position de l'objet sur le dessin ,

- la position de la sélection de l'objet. Pour montrer qu'un objet est sélectionné sur le dessin, un bord est tracé sur l'objet. Si ce bord est déplacé ou redimensionné, cela signifie que l'utilisateur veut déplacer ou redimensionner l'objet.

Les principales méthodes de cette classe permettent d'initialiser, de consulter ou de modifier ses données. D'autres méthodes permettent de

- tracer l'objet sur le dessin,

- déplacer l'objet sur le dessin,

- tracer la sélection de l'objet sur le dessin,

- déplacer la sélection de l'objet sur le dessin,

- redimensionner la sélection de l'objet,

- redimensionner l'objet ,

- modifier les chaînes de caractères de l'objet,

- demander à l'objet si une position lui appartient.

EltSimple :

Classe de base pour les éléments graphiques manipulés par le logiciel EGOO. Elle ne possède aucune donnée.

EltFixe :

Classe définissant un élément fixe d'un noeud ou d'un arc. Elle possède comme données :

- un objet de classe EltSimple représentant l'élément graphique de l'élément fixe,
- une référence sur un objet de classe RessEltFixe définissant la ressource de l'élément fixe.

La classe possède des méthodes d'accès et d'initialisation de ses données. En outre, les méthodes suivantes (définies dans la classe Element) ont été redéfinies :

- tracer l'objet sur le dessin,
- déplacer l'objet sur le dessin,
- redimensionner l'objet,
- modifier les chaînes de caractères de l'objet,
- demander à l'objet si une position lui appartient.

EltGraphe :

Classe de base pour les classes EltNoeud et EltArc. Elle possède les données communes à ces deux classes, à savoir :

- l'identifiant du noeud ou de l'arc,
- une liste d'objets de classe EltFixe représentant la liste des éléments fixes du noeud ou de l'arc,
- une liste de références d'objets de classe EltArc représentant les arcs qui portent sur l'objet en question.

Les méthodes associées à cette classe sont des méthodes d'initialisation, de modification et de consultation de ses données.

EltNoeud :

Classe représentant un noeud du dessin d'un schéma. Elle possède comme données :

- une liste d'objets de classe EltSimple représentant la liste des éléments graphiques définissant le contour du noeud,
- une liste de points d'ancrage du noeud ,
- une liste d'objets de classe EltGraphe représentant la liste des noeuds et arcs qui sont les composants admissibles du noeud en question,
- une référence vers un objet de classe RessNoeud représentant le noeud défini dans le fichier de ressources et étant le type de noeud de l'objet en question.

La classe possède des méthodes d'accès, d'initialisation et de modification de ses données. En outre, les méthodes suivantes (définies dans la classe Element) ont été redéfinies :

- tracer l'objet sur le dessin,
- déplacer l'objet sur le dessin,
- redimensionner l'objet,
- modifier les chaînes de caractères de l'objet,
- demander à l'objet si une position lui appartient.

EltArc :

Classe représentant un arc du dessin d'un schéma. Elle possède comme données :

- un objet de classe EltSimple représentant un élément graphique définissant le contour de l'arc,
- une référence vers un objet de classe RessArc représentant l'arc défini dans le fichier de ressources et étant le type d'arc de l'objet en question,
- une liste de références d'objets de classe EltGraphe. Cette liste représente la liste des noeuds et arcs sur lesquels l'arc en question porte.

La classe possède des méthodes d'accès, d'initialisation et de modification de ses données. En outre, les méthodes suivantes (définies dans la classe Element) ont été redéfinies :

- tracer l'objet sur le dessin,
- déplacer l'objet sur le dessin,
- redimensionner l'objet,
- modifier les chaînes de caractères de l'objet,
- demander à l'objet si une position lui appartient.

ListeElement :

Classe définissant une liste de références vers des objets de classe Element. Elle possède comme donnée une liste de références. Les méthodes suivantes (définies dans la classe Element) ont été redéfinies :

- tracer l'objet sur le dessin,
- déplacer l'objet sur le dessin,
- redimensionner l'objet,
- modifier les chaînes de caractères de l'objet,
- demander à l'objet si une position lui appartient.

Par la suite, nous allons décrire toutes les classes définissant un élément graphique. Pour ne pas alourdir inutilement le texte, nous n'allons pas spécifier les données et méthodes de chaque classe (sauf pour certains cas particuliers). Mais, il faut savoir que les méthodes suivantes (définies dans la classe Element) ont été redéfinies :

- tracer l'objet sur le dessin,
- déplacer l'objet sur le dessin,
- redimensionner l'objet,
- modifier les chaînes de caractères de l'objet,
- demander à l'objet si une position lui appartient.

EltDroiteComplexe :

Classe définissant une droite complexe. Elle possède comme données :

- une liste de listes d'objets de classe EltSimple définissant les groupes d'éléments graphiques constituant la droite complexe,
- un objet de classe EltSimple définissant le contour de la droite complexe,
- une référence vers un objet de classe RessDroite donnant la droite complexe définie dans le fichier de ressources,
- les contraintes de chaque groupe d'éléments graphiques.

EltDroite :

Classe définissant un segment de droite.

EltRectangle :

Classe définissant un rectangle.

EltTexte :

Classe définissant une chaîne de caractères statique ou dynamique.

EltEllipse :

Classe définissant une ellipse.

EltListebox :

Classe définissant une suite de chaînes de caractères.

EltPolygone :

Classe définissant un polygone.

EltMultiDroite :

Classe définissant une multi-droite.

4.6. Quelques techniques utilisées :

Dans cette partie, nous allons décrire quelques techniques utilisées pour l'implantation de l'éditeur graphique.

Une première technique utilisée concerne la sélection d'objets. Sur le dessin d'un schéma, l'utilisateur doit pouvoir sélectionner les noeuds, arcs et éléments fixes de noeuds et d'arcs. La sélection d'éléments graphiques est réalisée de la manière suivante. L'utilisateur ayant cliqué sur un élément graphique à l'aide de la souris, un parcours des noeuds, arcs et éléments fixes du dessin est réalisé. A chaque élément, il est demandé si la position désignée par l'utilisateur appartient ou non à l'élément. Dès qu'un noeud, arc ou élément fixe répond affirmativement à cette question, la recherche est arrêtée et cet élément est désigné comme élément sélectionné.

Une deuxième technique est l'utilisation d'un superviseur. Pour pouvoir maintenir des "variables globales" du programme, un objet nommé superviseur est utilisé. Ce superviseur possède toute l'information globale que n'importe quel objet peut demander. Le superviseur maintient, entre autre, comme information une référence sur l'objet fenêtre qui est la fenêtre active à un moment donné (la fenêtre active étant la fenêtre sur laquelle l'utilisateur est en train de travailler), l'objet couramment sélectionné, la liste des schémas maintenus par EGOO, ...

Chapitre 5

5. Conclusion

Dans ce chapitre, nous allons dans un premier temps exprimer quelques réflexions sur l'orienté objet. Par la suite, nous exposerons les conclusions sur le travail réalisé. Enfin, nous donnerons quelques perspectives intéressantes du logiciel EGOO.

5.1. Réflexions sur l'orienté objet

Cette partie expose l'évaluation de l'expérimentation de l'approche orientée objet que nous pouvons faire après avoir réalisé l'éditeur graphique EGOO. Cette évaluation est réalisée du point de vue de l'implantation et de la conception orienté objet.

Un des grands avantages, qui est couramment cité dans l'utilisation de l'orienté objet, est la réutilisation du code. Or, les classes qui sont définies dans l'implantation de l'éditeur graphique sont fortement liées les unes aux autres. En effet, prenons par exemple la classe EltNoeud définissant un noeud. Cette classe référence des objets de la classe EltArc, EltSimple, Schema, ... Une réutilisation de la classe EltNoeud, et par conséquent de son code, pour en définir une nouvelle classe dans un autre programme, entraîne l'utilisation de toutes les classes qui sont référencées par la classe EltNoeud. Une réutilisation de cette classe seule dans un autre programme est donc utopique.

La réutilisation d'une classe contenant des références sur des objets d'autres classes entraîne donc la réutilisation de toutes les classes d'objets référencés. Les classes qui sont le plus facilement réutilisables sont donc les classes n'ayant aucune référence sur d'autres objets; autrement dit, des classes autonomes. De telles classes sont, par exemple, des classes définissant une liste d'entiers, une liste de chaînes de caractères, un tableau, ... Cette perspective de réutilisation de code dans des architectures orientées objet est donc très limitée en ce qui concerne des classes spécifiques de cette architecture.

Lors de l'élaboration d'un programme orienté objet, deux parties importantes se distinguent. D'une part nous avons la conception de l'architecture du programme; d'autre part nous avons l'implantation de cette architecture. Si nous voulons élaborer un programme orienté objet, l'architecture de celui-ci doit être réalisée en orienté objet. Mais l'implantation de ce programme peut être réalisée dans un langage orienté objet ou non orienté objet.

Dans le cas où l'implantation est réalisée dans un langage non orienté objet, une certaine rigueur est demandée pour que cette implantation n'ait pas une structure complètement différente de celle qui a été définie dans l'architecture du programme.

Par contre, si un langage orienté objet est utilisé pour implanter le programme, la rigueur et la structure sont données par ce langage. En effet, la structure de l'implantation du programme est calquée sur l'architecture. Chaque classe définie dans l'implantation doit être une classe de l'architecture. La rigueur est donnée par le langage, car, ayant défini les classes du programme, le programmeur doit se soumettre à la structure mise en place. Il ne peut donc pas essayer de contourner cette structure.

Une dernière réflexion sur l'orienté objet porte sur la manière de penser et de réaliser l'implantation d'une classe, ou plus exactement l'implantation d'une méthode. On peut comparer l'implantation d'une méthode d'une classe à l'implantation d'une fonction récursive dans un langage non orienté objet. Prenons, par exemple, l'implantation d'une fonction récursive de parcours d'un arbre binaire. Lors de cette implantation, nous devons réfléchir sur une cellule quelconque de l'arbre et tout ce que nous devons "regarder", c'est que cette cellule comporte un sous-arbre droit et un sous-arbre gauche. Tout le reste de la structure de l'arbre n'est pas pris en compte.

De la même manière, lors de l'implantation d'une méthode d'une classe, nous ne devons nous soucier que de la classe sur laquelle nous travaillons. Lors de l'implantation, nous ne devons nous préoccuper que de l'influence de cette méthode sur les objets référencés ou appartenant à l'objet en question. Pour illustrer cela, prenons pour exemple la méthode Tracer d'un objet noeud de l'éditeur graphique. Pour tracer le noeud, nous ne nous préoccupons que du noeud et pas de toute la structure qui entoure ce noeud. Le noeud possède comme objets : une liste d'éléments graphiques définissant son contour, une liste d'éléments fixes et une liste de composants admissibles. Ces listes sont des objets qui ont chacun une représentation graphique (celle de tous les éléments qui les composent). La représentation graphique du noeud est donc celle des trois listes précédemment citées. Par conséquent, pour tracer le noeud, il suffit simplement de demander aux objets précédemment cités de se tracer.

5.2. Conclusions :

Le logiciel EGOO réalisé dans le cadre de ce travail n'a pu être complètement implanté. Les fonctionnalités de l'éditeur graphique implanté sont les suivantes :

- création et suppression d'un schéma,

- création, suppression, déplacement, modification, redimensionnement de noeuds et arcs du dessin d'un schéma,
- déplacement, modification, redimensionnement d'éléments fixes de noeuds et d'arcs,
- création de composants admissibles de noeuds,
- gestion du multi-fenêtrage,
- communication entre l'éditeur graphique et le serveur dans le sens EGOO vers serveur,
- sauvetage sur disque des schémas,
- déplacement de la vue d'une fenêtre.

La puissance du logiciel réalisé n'a pu être testée car il n'a été distribué à aucun concepteur de systèmes d'informations ou de logiciels. Néanmoins, nous avons pu représenter sans difficulté des exemples de représentations graphiques des modèles qui nous ont servi de base pour l'étude du logiciel, à savoir, les modèles E/A, le modèle de diagramme des flux, le modèle MAG, ...

L'implantation du logiciel EGOO a été facilitée par l'environnement MS-Windows. Mis à part un temps d'apprentissage relativement long, l'environnement MS-Windows nous a offert un grand nombre de fonctionnalités : gestion du multi-fenêtrage, gestion de la souris, gestion des boîtes de dialogue, gestion des menus, ... Ces fonctionnalités nous ont permis de nous concentrer sur la réalisation du logiciel EGOO sans avoir à se préoccuper de la réalisation de toutes ces fonctionnalités de bas niveau de l'interface. Nous regrettons cependant le manque de richesses des fonctions de MS-Windows en ce qui concerne le graphisme. En effet, MS-Windows ne fournit pas de fonctions pour tracer du texte dans n'importe quelle position (texte incliné par exemple à 45°, texte vertical, ...), pour effectuer une rotation du repère dans une fenêtre, ...

La méthode que nous avons utilisée dans la réalisation de l'implantation du logiciel est la méthode de la spirale. Nous avons tout d'abord construit un noyau du programme en implantant un nombre restreint de classes et de méthodes. A partir de cela, nous avons ajouté petit à petit des fonctionnalités au noyau du programme, en implantant de nouvelles classes et

de nouvelles méthodes. Ainsi, le logiciel EGOO a été réalisé en assemblant au fur et à mesure ses différents composants à un noyau exécutable.

Cette méthode de la spirale nous a permis de tester, à chaque ajout de fonctionnalité, si ce qui avait été implanté donnait le résultat escompté.

Le langage orienté objet C++ nous a facilité la tâche pour la réalisation de l'implantation du logiciel avec la méthode de la spirale. En effet, grâce au polymorphisme et aux liens dynamiques, nous pouvions ajouter une méthode à une classe, ou ajouter une classe entière sans devoir parcourir à nouveau tout le code pour modifier certaines parties tenant compte des classes d'objets utilisées.

Comme nous l'avons exposé dans l'introduction de ce travail, l'interaction entre le logiciel EGOO et MS-Windows a été réalisée en C et non en C++. Ce type d'interaction ne nous a posé aucun problème lors de l'implantation car le langage C++ permet l'appel de fonction C à partir de méthodes de classes C++ et inversement. Cependant, nous avons dû définir un ensemble de classes pour permettre de considérer une partie de l'environnement MS-Windows comme orienté objet. Ces classes définissent par exemple une fenêtre, une droite, un polygone, ... Ce travail aurait pu être évité si nous avions utilisé une hiérarchie de classes représentant MS-Windows "orienté objet". En effet, grâce à cette hiérarchie de classes pré définies, nous aurions simplement défini des classes spécifiques à l'éditeur graphique héritant de cette hiérarchie. Ainsi, seuls les comportements des classes propre à l'éditeur graphique auraient dû être implantées.

Pour illustrer cela, définissons, par exemple la classe représentant une droite du logiciel EGOO (notée EDroite), et supposons qu'il existe une classe pré définie représentant une droite (notée PDroite). Pour définir la classe EDroite, nous devons simplement spécifier qu'elle hérite de la classe PDroite. A cette classe, nous devons ajouter des méthodes spécifiques de l'éditeur graphique. A savoir, des méthodes pour demander si une position appartient à la droite, pour obtenir le point de la droite le plus proche d'un autre point, ...

L'utilisation d'une hiérarchie de classes, définissant un MS-Windows "orienté objet", permet donc une économie de temps et de code. De plus, nous pouvons, à ce moment là, travailler complètement en orienté objet et non en "mélangeant" du code C++ avec du code C.

5.3. Perspectives :

Une première perspective d'avenir concerne les fonctionnalités du logiciel devant encore être implantées. Ces fonctionnalités sont les suivantes :

- gestion des sous-diagrammes,
- gestion des déplacements d'un noeud ou arc d'un noeud père vers un autre noeud père,
- gestion du zoom sur les fenêtres d'un schéma,
- gestion des râteaux comme éléments graphiques statiques,
- gestion des requêtes émises par le serveur à l'éditeur graphique.

Nous pouvons également ajouter certaines fonctionnalités comme l'impression des schémas, la possibilité de " undo ", ...

Une autre perspective du logiciel EGOO concerne ses performances. Les performances du logiciel devraient être étudiées au niveau de l'interaction entre le logiciel et MS-Windows, au niveau du fichier de ressources et au niveau du code C++. En ce qui concerne l'interaction entre le logiciel et MS-Windows, une étude devrait être réalisée afin d'optimiser les appels de fonctions de MS-Windows. Cela permettrait d'augmenter la rapidité d'affichage des dessins, ainsi que la rapidité de déplacement d'éléments, de déplacement de la vue d'une fenêtre, ... Au niveau du fichier de ressources, une optimisation devrait être réalisée afin d'augmenter la rapidité de sa lecture. Une telle optimisation consisterait par exemple en l'utilisation d'un fichier de ressources " compilé " au lieu d'utiliser une version du fichier de ressources sous forme de texte ASCII. Enfin, un test de performance devrait être réalisé au niveau du code C++ afin d'augmenter la rapidité d'exécution.

L'élaboration d'un fichier de ressources étant un travail précis et fastidieux, il serait utile d'utiliser un éditeur graphique pour le construire. Cet éditeur graphique permettrait de construire tous les noeuds, arcs et droites complexes d'un type de schéma, en définissant graphiquement ces éléments. Par exemple, pour la définition d'un noeud, l'utilisateur aurait à sa disposition des éléments graphiques qu'il utiliserait pour construire le contour du noeud. Sur ce contour, il définirait également les éléments fixes ainsi que toutes leurs contraintes, les points d'ancrage, ...

Cet éditeur graphique pourrait même faire partie intégrante du logiciel EGOO. En effet, le logiciel EGOO, possédant déjà des fonctions d'éditeur graphique, pourrait construire un schéma particulier qui décrirait les ressources d'un type de schéma.

Le logiciel EGOO étant général et basé sur une architecture client-serveur, il pourrait être " facilement " intégré dans un outil CASE, il suffit de définir les fichiers de ressources ainsi que de réaliser le serveur du logiciel EGOO. Pour cela, deux types d'outils CASE doivent être distingués : les outils CASE spécifiques et les outils CASE génériques. Les outils CASE spécifiques sont des outils CASE ne travaillant que sur un nombre limité de modèles connus d'avance. Par contre, les outils CASE génériques travaillent sur un modèle général qui peut être instancié à divers modèles.

Traitons le cas de l'intégration du logiciel EGOO dans un outil CASE spécifique. Cet outil CASE ne travaillant que sur un nombre limité de modèles, il suffit de créer pour chaque modèle un fichier de ressources le décrivant. Il faudra également concevoir et réaliser un serveur respectant le protocole de communication avec le logiciel EGOO, qui pourra maintenir les graphes sémantiques correspondant aux schémas traités par EGOO.

Dans le cas de l'intégration du logiciel EGOO dans un outil CASE générique, il faudra procéder différemment. Cet outil doit disposer d'une méta-graphique qui permet de décrire tous les éléments graphiques des modèles qu'il peut gérer. Pour chaque modèle qu'il doit traiter, il doit pouvoir générer, à partir de sa méta-graphique, le fichier de ressources d'EGOO correspondant à la représentation graphique du modèle. L'outil doit également disposer d'un serveur de EGOO qui doit être général afin de pouvoir traiter tous les modèles qu'il peut représenter.

Une dernière perspective du logiciel EGOO concerne la cohérence du fichier de ressources. En effet, dans la version actuelle, aucun contrôle de cohérence n'est réalisé. Tout fichier de ressources donné au logiciel EGOO est considéré comme cohérent. Un contrôle de cohérence devrait être réalisé lors de la lecture du fichier de ressources.

Chapitre 6

6. Bibliographie

[1] F. Bodart et Y. Pigneur, *Conception assistée des systèmes d'information : Etude d'opportunité et analyse conceptuelle*, Ed. Masson, 1989.

[2] J.-L. Hainaut, *Conception assistée des systèmes d'information : Conception de la base de données*, Ed. Masson, 1986.

[3] Stanley B. Lippman, *C++ primer*, Addison-Wesley Publishing Compagny, Reading, USA, 1989.

[4] Tim Korson & John D. McGragor, *Understanding Object-oriented : A Unifying Paradigm*, Communication of the ACM, vol. 33, n° 9, septembre 1990.

[5] David Jordan, *Implementation Benefits of C++ Langage Mechanisms*, Communication of the ACM, vol. 33, n° 9, septembre 1990.

[6] K. Gorlen, S. Orlow, P. Plexico, *Conception et programmation orientées objet en C++*, Ed. Dunod Informatique.

[7] Jeffrey M. Richter, *Windows 3 : A developer's Guide*, M&T Publishing, Inc. 1991.

[8] Peter Wilken & Dirk Honekamp, *Windows System Programming*, Abacus.

[9] Charles Petzold, *Programming Windows*, Microsoft Press, 1990.

[10] Microsoft Corporation, *Windows Programmer's Reference*, Microsoft Press,1990.

[11] Microsoft Corporation, *Windows Programming Tools*, Microsoft Press,1990.

[12] J.D. Foley & A. Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Compagny, Reading, USA, 1982.

[13] Joëlle Coutaz, *Abstractions pour Interfaces Interactives*, Technique et Science Informatique, vol. 5, n° 4, 1986.

Annexe

Annexe

Cette annexe décrit les fichier de ressources du modèle E/A et du modèle du dyagramme des flux.

Fichier de ressources du modèle E/A

```
% Le schéma décrit ci-dessous est celui qui correspond au schéma E/A.

SCHEMA ERA


% Droite de contrainte d'inclusion

% Il s'agit d'une multi-droite, disposant d'une flèche à l'une de ses
% extrémités, sur laquelle est venu se coller un cercle contenant la
% lettre I ( synonyme de la contrainte d'inclusion ).

DROITE CI
 BEGIN
  MULTIDROITE 2 0 50 200 50 SOLID 1 0 0 0 FLECHED
  ANCRAGE = 0 50 200 50
  BEGIN
    ROTATE = YES
    CONTRAINTE = 0 0
    ELLIPSE 50 0 150 100 SOLID 1 0 0 0 SOLID 255 255 255
    TEXTE 75 5 125 95 STATIC 90 CENTER 0 0 0 TRANSPARENT 0 0 0 I
  END
 END

% Droite de contrainte d'exclusion

% Il s'agit d'une multi-droite sur laquelle est venu se coller un cercle
% contenant le caractère # ( synonyme de la contrainte d'exclusion ).

DROITE CEX
 BEGIN
  MULTIDROITE 2 0 50 200 50 SOLID 1 0 0 0 SIMPLE
  ANCRAGE = 0 50 200 50
  BEGIN
    ROTATE = YES
    CONTRAINTE = 0 0
    ELLIPSE 50 0 150 100 SOLID 1 0 0 0 SOLID 255 255 255
    TEXTE 75 5 125 95 STATIC 90 CENTER 0 0 0 TRANSPARENT 0 0 0 #
  END
```

```
 END

% Droite de contrainte d'égalité

% Il s'agit d'une multi-droite sur laquelle est venu se coller un cercle
% contenant la lettre E ( synonyme de la contrainte d'exclusion ).

DROITE  CEG
 BEGIN
  MULTIDROITE 2 0  50   200  50  SOLID  1    0 0 0  SIMPLE
  ANCRAGE = 0 50 200 50
  BEGIN
    ROTATE = YES
    CONTRAINTE = 0 0
    ELLIPSE 50 0 150 100 SOLID  1   0 0 0  SOLID 255 255 255
    TEXTE 75 5  125 95 STATIC 90 CENTER 0 0 0 TRANSPARENT 0 0 0 =
  END
 END

% Définition du noeud Association

% Il s'agit d'un noeud dont le contour est constitué d'un hexagone
% et d'une droite dans sa partie supérieure. Les composants fixes sont
% d'une part le nom de l'association et d'autre part les noms des
% attributs. Ce noeud ne possède pas de composant admissible. Le nom
% de l'association tient sur deux lignes.

NOEUD Association
 BEGIN
  CONTOUR
    TYPE_MOVE_COMP_ADM = ALL
    TYPE_RESIZE_COMP_ADM = ALL
    POSS_MOVE = UNLIMITED
    MODIF   : FCT = FctModif
    NAVIG   : FCT = FctNavig
    MOVE    : FCT = NULL
    INFO : FCT = FctInfo
    DELETE : FCT = FctDelete
    INSERT : FCT = FctInsert
    ENCRAGE = 275 0 550 165 275 330 0 165
    POLYGONE 6 50 0 500 0 550 165 500 330 50 330 0 165 SOLID  1   0 0 0  ALTERNATE SOLID
255 255 255
    DROITE    15 130 535 130 SOLID  1   0 0 0  SIMPLE
  COMPOSANTS_FIXES
    NomAssociation
    TEXTE 60 10 490 120 VARIABLE 50 CENTER 0 0 0 TRANSPARENT 0 0 0 Association
    TYPE_SENS_RESIZE =
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL = HORI
```

```
    POSS_MOVE_RESIZE_GLOBAL = VERTHORI
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif

    NomsAttributs
    LISTEBOX 60 140 490 320 50 LEFT  0 0 0 TRANSPARENT 0 0 0
    TYPE_SENS_RESIZE = VERT
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL = VERTHORI
    POSS_MOVE_RESIZE_GLOBAL = VERTHORI
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif
  COMPOSANTS_ADMISSIBLES
 END

% Définition du noeud Association

% Il s'agit d'un noeud dont le contour est constitué d'un hexagone
% et d'une droite dans sa partie supérieure. Les composants fixes sont
% d'une part le nom de l'association et d'autre part les noms des
% attributs. Ce noeud ne possède pas de composant admissible. Le nom
% de l'association tient sur une seule ligne.

NOEUD Association_1L
 BEGIN
  CONTOUR
    TYPE_MOVE_COMP_ADM = ALL
    TYPE_RESIZE_COMP_ADM = ALL
    POSS_MOVE = UNLIMITED
    MODIF   : FCT = FctModif
    NAVIG   : FCT = FctNavig
    MOVE   : FCT = NULL
    INFO : FCT = FctInfo
    DELETE : FCT = FctDelete
    INSERT : FCT = FctInsert
    ENCRAGE = 275 0 550 165 275 330 0 165
    POLYGONE 6 50 0 500 0 550 165 500 330 50 330 0 165 SOLID  1   0 0 0  ALTERNATE SOLID
255 255 255
    DROITE   25 70 525 70 SOLID  1   0 0 0  SIMPLE
  COMPOSANTS_FIXES
    NomAssociation
    TEXTE 60 10 490 60 VARIABLE 50 CENTER 0 0 0 TRANSPARENT 0 0 0 Association
    TYPE_SENS_RESIZE =
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL = HORI
    POSS_MOVE_RESIZE_GLOBAL = VERTHORI
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif
```

```
    NomsAttributs
    LISTEBOX 60 80 490 320 50 LEFT  0 0 0 TRANSPARENT 0 0 0
    TYPE_SENS_RESIZE = VERT
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL = VERTHORI
    POSS_MOVE_RESIZE_GLOBAL = VERTHORI
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif
  COMPOSANTS_ADMISSIBLES
 END

% Définition du noeud Entite :

% Il s'agit d'un noeud dont le contour est constitué d'un rectangle.
% Les composants fixes sont le nom de l'entité, la droite dessinée dans
% la partie supérieure du contour et les noms des attributs. Ce noeud
% ne possède pas de composant admissible. Le noeud de l'entité tient
% sur deux lignes.

NOEUD Entite
 BEGIN
  CONTOUR
    TYPE_MOVE_COMP_ADM = ALL
    TYPE_RESIZE_COMP_ADM = ALL
    POSS_MOVE = UNLIMITED
    MODIF   : FCT = FctModif
    NAVIG   : FCT = FctNavig
    MOVE    : FCT = NULL
    INFO : FCT = FctInfo
    DELETE : FCT = FctDelete
    INSERT : FCT = FctInsert
    ENCRAGE = 225 0 450 195 225 390 0 195
    RECTANGLE 0 0 450 390 SOLID  1   0 0 0  SOLID 255 255 255
  COMPOSANTS_FIXES
    NomEntite
    TEXTE 10 10 440 120 VARIABLE 50 CENTER  0 0 0 TRANSPARENT 0 0 0  ENTITE
    TYPE_SENS_RESIZE = HORI
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL = HORI
    POSS_MOVE_RESIZE_GLOBAL = HORI
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif

    Droite
    DROITE  0  130 450 130  SOLID  1   0 0 0  SIMPLE
    TYPE_SENS_RESIZE =
```

```
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL = HORI
    POSS_MOVE_RESIZE_GLOBAL = HORI
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif

    NomsAttributs
    LISTEBOX 10 140 440 380 LEFT  0 0 0 TRANSPARENT 0 0 0
    TYPE_SENS_RESIZE = VERT
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL = VERTHORI
    POSS_MOVE_RESIZE_GLOBAL = HORI
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif
  COMPOSANTS_ADMISSIBLES
 END

% Définition du noeud Entite  :

% Il s'agit d'un noeud dont le contour est constitué d'un rectangle.
% Les composants fixes sont le nom de l'entité, la droite dessinée dans
% la partie supérieure du contour et les noms des attributs. Ce noeud
% ne possède pas de composant admissible. Le nom de l'entité doit tenir
% sur une seule ligne.

NOEUD Entite_1L
 BEGIN
  CONTOUR
    TYPE_MOVE_COMP_ADM = ALL
    TYPE_RESIZE_COMP_ADM = ALL
    POSS_MOVE = UNLIMITED
    MODIF   : FCT = FctModif
    NAVIG   : FCT = FctNavig
    MOVE   : FCT = NULL
    INFO : FCT = FctInfo
    DELETE : FCT = FctDelete
    INSERT : FCT = FctInsert
    ENCRAGE = 225 0 450 195 225 390 0 195
    RECTANGLE 0 0 450 390 SOLID  1   0 0 0  SOLID 255 255 255
  COMPOSANTS_FIXES
    NomEntite
    TEXTE 10 10 440  60 VARIABLE 50 CENTER  0 0 0 TRANSPARENT 0 0 0  ENTITE
    TYPE_SENS_RESIZE = HORI
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL = HORI
    POSS_MOVE_RESIZE_GLOBAL = HORI
```

```
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif

    Droite
    DROITE  0  70 450 70 SOLID  1   0 0 0  SIMPLE
    TYPE_SENS_RESIZE =
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL = HORI
    POSS_MOVE_RESIZE_GLOBAL = HORI
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif

    NomsAttributs
    LISTEBOX 10 80 440 380 LEFT  0 0 0 TRANSPARENT 0 0 0
    TYPE_SENS_RESIZE = VERT
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL = VERTHORI
    POSS_MOVE_RESIZE_GLOBAL = HORI
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif
  COMPOSANTS_ADMISSIBLES
 END


% Définition d'un arc Rôle

% Il s'agit d'un arc dont le contour est constitué d'une multi-droite.
% Les composants fixes sont le nom du rôle et la connectivité. Les noeuds
% sur lesquels peut porter l'arc sont les entités et les associations.

ARC  Role
 BEGIN
   CONTOUR
     MODIF   : FCT = FctModif
     INFO : FCT = FctInfo
     DELETE : FCT = FctDelete
     INSERT : FCT = FctInsert
     MULTIDROITE 2 0 100  500 100  SOLID  1    0 0 0  SIMPLE
   COMPOSANTS_FIXES
     Nom_du_Role
     TEXTE 20 0 200 60 VARIABLE 50 CENTER 0 0 0  TRANSPARENT 255 255 255
     TYPE_SENS_RESIZE = VERTHORI
     POSS_MOVE = YES
     POSS_MOVE_GLOBAL = VERTHORI
     TYPE_SENS_RESIZE_GLOBAL =
     POSS_MOVE_RESIZE_GLOBAL = VERTHORI
     CONTRAINTE = 0 0
```

```
    MODIF   : FCT = FctModif

    Connectivite
    TEXTE 20 140 200 200 VARIABLE 50 CENTER 0 0 0  TRANSPARENT 255 255 255
    TYPE_SENS_RESIZE = VERTHORI
    POSS_MOVE = YES
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL =
    POSS_MOVE_RESIZE_GLOBAL = VERTHORI
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif
  NOEUD_ARC_ADMISSIBLES
    Entite
    Entite_1L
    Association
    Association_1L
END

% Définition d'un arc Contrainte d'inclusion de rôles

% Il s'agit d'un arc dont le contour est constitué d'une droite de
% contrainte d'inclusion ( définie ci-dessus ). Cet arc ne possède pas
% de composant fixe. Les arcs sur lesquels peut porter cet arc de
% contrainte d'inclusion de rôles sont les arcs de type rôle et de type
% sous-type.

ARC  CI_de_Role
 BEGIN
  CONTOUR
    MODIF   : FCT = FctModif
    INFO : FCT = FctInfo
    DELETE : FCT = FctDelete
    INSERT : FCT = FctInsert
    CI  0  0   200  200
  COMPOSANTS_FIXES
  NOEUD_ARC_ADMISSIBLES
    Role
    Sous-Type
END

% Définition d'un arc Contrainte d'exclusion de rôles

% Il s'agit d'un arc dont le contour est constitué d'une droite de
% contrainte d'exclusion ( définie ci-dessus ). Cet arc ne possède pas
% de composant fixe. Les arcs sur lesquels peut porter cet arc de
% contrainte d'exclusion de rôles sont les arcs de type rôle et de type
% sous-type.

ARC  CEX_de_Role
```

```
BEGIN
  CONTOUR
    MODIF   : FCT = FctModif
    INFO : FCT = FctInfo
    DELETE : FCT = FctDelete
    INSERT : FCT = FctInsert
    CEX  0  0   200  200
  COMPOSANTS_FIXES
  NOEUD_ARC_ADMISSIBLES
    Role
    Sous-Type
END

% Définition d'un arc Contrainte d'égalité de rôles

% Il s'agit d'un arc dont le contour est constitué d'une droite de
% contrainte d'égalité ( définie ci-dessus ). Cet arc ne possède pas
% de composant fixe. Les arcs sur lesquels peut porter cet arc de
% contrainte d'égalité de rôles sont les arcs de type rôle.

ARC  CEG_de_Role
BEGIN
  CONTOUR
    MODIF   : FCT = FctModif
    INFO : FCT = FctInfo
    DELETE : FCT = FctDelete
    INSERT : FCT = FctInsert
    CEG  0  0   200  200
  COMPOSANTS_FIXES
  NOEUD_ARC_ADMISSIBLES
    Role
END

%  Définition d'un arc Contrainte d'inclusion d'associations

% Il s'agit d'un arc dont le contour est constitué d'une droite de
% contrainte d'inclusion ( définie ci-dessus ). Cet arc ne possède pas
% de composant fixe. Les noeuds sur lesquels peut porter cet arc sont
% les associations.

ARC  CI_de_Asso
BEGIN
  CONTOUR
    MODIF   : FCT = FctModif
    INFO : FCT = FctInfo
    DELETE : FCT = FctDelete
    INSERT : FCT = FctInsert
    CI  0  0   200  200
  COMPOSANTS_FIXES
```

```
   NOEUD_ARC_ADMISSIBLES
     Association
     Association_1L
 END

%  Définition d'un arc Contrainte d'exclusion d'associations


% Il s'agit d'un arc dont le contour est constitué d'une droite de
% contrainte d'exclusion ( définie ci-dessus ). Cet arc ne possède pas
% de composant fixe. Les noeuds sur lesquels peut porter cet arc sont
% les associations.

ARC  CEX_de_Asso
 BEGIN
   CONTOUR
     MODIF   : FCT = FctModif
     INFO : FCT = FctInfo
     DELETE : FCT = FctDelete
     INSERT : FCT = FctInsert
     CEX  0  0   200  200
   COMPOSANTS_FIXES
   NOEUD_ARC_ADMISSIBLES
     Association
     Association_1L
 END

% Définition d'un arc Contrainte d'égalité d'associations


% Il s'agit d'un arc dont le contour est constitué d'une droite de
% contrainte d'égalité ( définie ci-dessus ). Cet arc ne possède pas
% de composant fixe. Les noeuds sur lesquels peut porter cet arc sont
% les associations.

ARC  CEG_de_Asso
 BEGIN
   CONTOUR
     MODIF   : FCT = FctModif
     INFO : FCT = FctInfo
     DELETE : FCT = FctDelete
     INSERT : FCT = FctInsert
     CEG  0  0   200  200
   COMPOSANTS_FIXES
   NOEUD_ARC_ADMISSIBLES
     Association
     Association_1L
 END
```

```
% Définition d'un arc Sous-type

% Il s'agit d'un arc dont le contour est composé d'une multi-droite.
% Les composants fixes sont le nom du sous-type ainsi que deux
% contraintes de connectivité. Cet arc peut porter sur des entités.

ARC  Sous-Type
 BEGIN
   CONTOUR
     MODIF   : FCT = FctModif
     INFO : FCT = FctInfo
     DELETE : FCT = FctDelete
     INSERT : FCT = FctInsert
     MULTIDROITE 2 0 100  500 100  SOLID  1    0 0 0  FLECHED
   COMPOSANTS_FIXES
     Nom_du_Sous-Type
     TEXTE 20 0 200 40 VARIABLE 50 CENTER 0 0 0  TRANSPARENT 255 255 255
     TYPE_SENS_RESIZE = VERTHORI
     POSS_MOVE = YES
     POSS_MOVE_GLOBAL = VERTHORI
     TYPE_SENS_RESIZE_GLOBAL =
     POSS_MOVE_RESIZE_GLOBAL = VERTHORI
     CONTRAINTE = 0 0
     MODIF   : FCT = FctModif

     Connectivite1
     TEXTE 20 140 200 180 VARIABLE 50 CENTER 0 0 0  TRANSPARENT 255 255 255
     TYPE_SENS_RESIZE = VERTHORI
     POSS_MOVE = YES
     POSS_MOVE_GLOBAL = VERTHORI
     TYPE_SENS_RESIZE_GLOBAL =
     POSS_MOVE_RESIZE_GLOBAL = VERTHORI
     CONTRAINTE = 0 0
     MODIF   : FCT = FctModif

     Connectivite2
     TEXTE 320 140 500 180 VARIABLE 50 CENTER 0 0 0  TRANSPARENT 255 255 255
     TYPE_SENS_RESIZE = VERTHORI
     POSS_MOVE = YES
     POSS_MOVE_GLOBAL = VERTHORI
     TYPE_SENS_RESIZE_GLOBAL =
     POSS_MOVE_RESIZE_GLOBAL = VERTHORI
     CONTRAINTE = 0 0
     MODIF   : FCT = FctModif
   NOEUD_ARC_ADMISSIBLES
     Entite
     Entite_1L
END
```

```
% Définition d'un arc Sous-Type ( entre association et entité )

% Il s'agit d'un arc dont le contour est composé d'une multi-droite.
% Les composants fixes sont le nom du sous-type ainsi que la connectivité.
% Cet arc peut porter sur des entités et des associations.

ARC  Sous-Type_Ass_Ent
 BEGIN
   CONTOUR
     MODIF   : FCT = FctModif
     INFO : FCT = FctInfo
     DELETE : FCT = FctDelete
     INSERT : FCT = FctInsert
     MULTIDROITE 2 0 100  500 100  SOLID  1    0 0 0  FLECHED
   COMPOSANTS_FIXES
     Nom_du_Sous-Type
     TEXTE 20 0 200 40 VARIABLE 50 CENTER 0 0 0  TRANSPARENT 255 255 255
     TYPE_SENS_RESIZE = VERTHORI
     POSS_MOVE = YES
     POSS_MOVE_GLOBAL = VERTHORI
     TYPE_SENS_RESIZE_GLOBAL =
     POSS_MOVE_RESIZE_GLOBAL = VERTHORI
     CONTRAINTE = 0 0
     MODIF   : FCT = FctModif

     Connectivite
     TEXTE 20 140 200 180 VARIABLE 50 CENTER 0 0 0  TRANSPARENT 255 255 255
     TYPE_SENS_RESIZE = VERTHORI
     POSS_MOVE = YES
     POSS_MOVE_GLOBAL = VERTHORI
     TYPE_SENS_RESIZE_GLOBAL =
     POSS_MOVE_RESIZE_GLOBAL = VERTHORI
     CONTRAINTE = 0 0
     MODIF   : FCT = FctModif
   NOEUD_ARC_ADMISSIBLES
     Entite
     Entite_1L
     Association
     Association_1L
END
```

Fichier de ressources du modèle du diagramme des flux

```
% Le schéma décrit ci-dessous est celui qui correspond au schéma
% du diagramme de flux.

SCHEMA FLUX

% Définition du noeud Traitement

% Il s'agit d'un noeud dont le contour est constitué d'un rectangle.
% Le composant fixe est le nom du traitement. Ce noeud ne possède
% pas de composant admissible.

NOEUD Traitement
 BEGIN
  CONTOUR
    TYPE_MOVE_COMP_ADM = ALL
    TYPE_RESIZE_COMP_ADM = ALL
    POSS_MOVE = UNLIMITED
    MODIF   : FCT = FctModif
    NAVIG   : FCT = FctNavig
    MOVE    : FCT = FctMove
    INFO : FCT = FctInfo
    DELETE : FCT = FctDelete
    INSERT : FCT = FctInsert
    ENCRAGE = 225 0 450 65 225 130 0 65
    RECTANGLE 0 0 450 130 SOLID  1  0 0 0  SOLID 255 255 255
  COMPOSANTS_FIXES
     Nom
     TEXTE 10 10 440 120 VARIABLE 50 CENTER 0 0 0  OPAQUE 255 255 255 Texte
     TYPE_SENS_RESIZE =
     POSS_MOVE = NO
     POSS_MOVE_GLOBAL = VERTHORI
     TYPE_SENS_RESIZE_GLOBAL = VERTHORI
     POSS_MOVE_RESIZE_GLOBAL = VERTHORI
     CONTRAINTE = 0 0
     MODIF   : FCT = FctModif
  COMPOSANTS_ADMISSIBLES
 END

% Définition du noeud Message

% Il s'agit d'un noeud dont le contour est constitué d'un polygone.
% Le composant fixe est le nom du message. Ce noeud ne dispose pas
% de composant admissible.

NOEUD Message
 BEGIN
  CONTOUR
```

```
    TYPE_MOVE_COMP_ADM = ALL
    TYPE_RESIZE_COMP_ADM = ALL
    POSS_MOVE = UNLIMITED
    MODIF   : FCT = FctModif
    NAVIG   : FCT = FctNavig
    MOVE    : FCT = FctMove
    INFO : FCT = FctInfo
    DELETE : FCT = FctDelete
    INSERT : FCT = FctInsert
    ENCRAGE = 225 0 450 65 225 130 225 172 0 65
    POLYGONE 12 0 0 450 0 450 130 400 155 350 160 300 165 250 170 200 175 150 180 100 190
50 205 0 230 SOLID  1   0 0 0  ALTERNATE SOLID 255 255 255
  COMPOSANTS_FIXES
    Nom
    TEXTE 10 10 440 120 VARIABLE 50 CENTER 0 0 0  OPAQUE 255 255 255 Texte
    TYPE_SENS_RESIZE =
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL = VERTHORI
    POSS_MOVE_RESIZE_GLOBAL = VERTHORI
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif
  COMPOSANTS_ADMISSIBLES
 END

% Définition du noeud Message Interne

% Il s'agit d'un noeud dont le contour est constitué d'un polygone.
% Le composant fixe est le nom du message interne. Ce noeud ne dispose
% pas de composant admissible.

NOEUD Message_Int
 BEGIN
  CONTOUR
    TYPE_MOVE_COMP_ADM = ALL
    TYPE_RESIZE_COMP_ADM = ALL
    POSS_MOVE = UNLIMITED
    MODIF   : FCT = FctModif
    NAVIG   : FCT = FctNavig
    MOVE    : FCT = FctMove
    INFO : FCT = FctInfo
    DELETE : FCT = FctDelete
    INSERT : FCT = FctInsert
    ENCRAGE = 225 0 450 65 225 130 225 172 0 65
    POLYGONE 12 0 0 450 0 450 130 400 155 350 160 300 165 250 170 200 175 150 180 100 190
50 205 0 230 DOT  1   0 0 0  ALTERNATE SOLID 255 255 255
  COMPOSANTS_FIXES
    Nom
    TEXTE 10 10 440 120 VARIABLE 50 CENTER 0 0 0  OPAQUE 255 255 255 Texte
    TYPE_SENS_RESIZE =
```

```
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL = VERTHORI
    POSS_MOVE_RESIZE_GLOBAL = VERTHORI
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif
  COMPOSANTS_ADMISSIBLES
 END

% Définition d'une unité organisationnelle

% Il s'agit d'un noeud dont le contour est constitué d'un rectangle.
% Les composants fixes sont le nom de l'unité organisationnelle et la
% droite dessinée dans sa partie supérieure. Ce noeud dispose des
% composants admissibles suivants :  Traitement, Message, Message_Int,
% Phase_Auto, Phase_Manuelle, Phase_Interactive, Memoire, Arc

NOEUD Unite_orga
 BEGIN
  CONTOUR
    TYPE_MOVE_COMP_ADM = ALL
    TYPE_RESIZE_COMP_ADM = ALL
    POSS_MOVE = UNLIMITED
    MODIF   : FCT = FctModif
    NAVIG   : FCT = FctNavig
    MOVE   : FCT = NULL
    INFO : FCT = FctInfo
    DELETE : FCT = FctDelete
    INSERT : FCT = FctInsert
    ENCRAGE = 500 0
    RECTANGLE 0 0 1000 8000 SOLID  1   0 0 0  SOLID 255 255 255
  COMPOSANTS_FIXES
    NomEntite
    TEXTE 10 10 440 120 VARIABLE 50 CENTER  0 0 0 TRANSPARENT 0 0 0  ENTITE
    TYPE_SENS_RESIZE = HORI
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL = HORI
    POSS_MOVE_RESIZE_GLOBAL = HORI
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif

    Droite
    DROITE  0  130   1000  130  SOLID  1   0 0 0  SIMPLE
    TYPE_SENS_RESIZE =
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL = HORI
    POSS_MOVE_RESIZE_GLOBAL = HORI
```

```
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif
  COMPOSANTS_ADMISSIBLES
    Traitement
    Message
    Message_Int
    Phase_Auto
    Phase_Manuelle
    Phase_Interactive
    Memoire
    Arc
 END

% Définition d'une phase automatique

% Il s'agit d'un noeud dont le contour est constitué d'un double
% rectangle. Le composant fixe est le nom de la phase automatique.
% Ce noeud ne dispose pas de composant admissible.

NOEUD Phase_Auto
 BEGIN
  CONTOUR
    TYPE_MOVE_COMP_ADM = ALL
    TYPE_RESIZE_COMP_ADM = ALL
    POSS_MOVE = UNLIMITED
    MODIF   : FCT = FctModif
    NAVIG   : FCT = FctNavig
    MOVE   : FCT = FctMove
    INFO : FCT = FctInfo
    DELETE : FCT = FctDelete
    INSERT : FCT = FctInsert
    ENCRAGE = 50 0 50 100 100 50 0 50
    RECTANGLE 0 0 460 140 SOLID  1  0 0 0  SOLID 255 255 255
    RECTANGLE 5 5 455 135 SOLID  1  0 0 0  SOLID 255 255 255
  COMPOSANTS_FIXES
    Nom
    TEXTE 15 15 445 125 VARIABLE 50 CENTER 0 0 0  OPAQUE 255 255 255 Texte
    TYPE_SENS_RESIZE =
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL = VERTHORI
    POSS_MOVE_RESIZE_GLOBAL = VERTHORI
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif
  COMPOSANTS_ADMISSIBLES
 END

% Définition d'un noeud Phase manuelle
```

```
% Il s'agit d'un noeud dont le contour est constitué d'un rectangle.
% Les composants fixes sont le nom de la phase manuelle, deux segments
% de droite entourant le temps de réalisation de la tâche manuelle. Ce
% noeud ne dispose pas de composant admissible.

NOEUD Phase_Manuelle
 BEGIN
  CONTOUR
    TYPE_MOVE_COMP_ADM = ALL
    TYPE_RESIZE_COMP_ADM = ALL
    POSS_MOVE = UNLIMITED
    MODIF   : FCT = FctModif
    NAVIG   : FCT = FctNavig
    MOVE    : FCT = FctMove
    INFO : FCT = FctInfo
    DELETE : FCT = FctDelete
    INSERT : FCT = FctInsert
    ENCRAGE = 255 0   0 65 510 65 255 130
    RECTANGLE 0 0 510 130 SOLID  1   0 0 0  SOLID 255 255 255
  COMPOSANTS_FIXES
    Nom
    TEXTE 10 10 440 120 VARIABLE 50 CENTER  0 0 0 TRANSPARENT 0 0 0  ENTITE
    TYPE_SENS_RESIZE = HORI
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL = HORI
    POSS_MOVE_RESIZE_GLOBAL = HORI
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif

    Droite1
    DROITE  450  70   510  70  SOLID  1   0 0 0  SIMPLE
    TYPE_SENS_RESIZE =
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL =
    POSS_MOVE_RESIZE_GLOBAL =
    CONTRAINTE = 510 130
    MODIF   : FCT = FctModif

    Droite2
    DROITE   450  70 450  130  SOLID  1   0 0 0  SIMPLE
    TYPE_SENS_RESIZE =
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL =
    POSS_MOVE_RESIZE_GLOBAL =
    CONTRAINTE = 510 130
    MODIF   : FCT = FctModif
```

```
    Temps
    TEXTE 460 80  510 120 VARIABLE 50 CENTER  0 0 0 TRANSPARENT 0 0 0  ENTITE
    TYPE_SENS_RESIZE =
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL =
    POSS_MOVE_RESIZE_GLOBAL =
    CONTRAINTE = 510 130
    MODIF   : FCT = FctModif
  COMPOSANTS_ADMISSIBLES
 END

% Définition du noeud Phase interactive

% Il s'agit d'un noeud dont le contour est constitué d'un trapèze.
% Les composants fixes sont le nom de la phase interactive, deux segments
% de droite entourant le temps de réalisation de la tâche manuelle. Ce
% noeud ne dispose pas de composant admissible.


NOEUD Phase_Interactive
 BEGIN
  CONTOUR
    TYPE_MOVE_COMP_ADM = ALL
    TYPE_RESIZE_COMP_ADM = ALL
    POSS_MOVE = UNLIMITED
    MODIF   : FCT = FctModif
    NAVIG   : FCT = FctNavig
    MOVE   : FCT = FctMove
    INFO : FCT = FctInfo
    DELETE : FCT = FctDelete
    INSERT : FCT = FctInsert
    ENCRAGE = 255 12   0 77 510 77 255 155
    POLYGONE 4 0 25 510 0 510 155 0 155 SOLID  1   0 0 0  ALTERNATE SOLID 255 255 255
  COMPOSANTS_FIXES
    Nom
    TEXTE 10 35 500 145 VARIABLE 50 CENTER  0 0 0 TRANSPARENT 0 0 0  ENTITE
    TYPE_SENS_RESIZE = HORI
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL = HORI
    POSS_MOVE_RESIZE_GLOBAL = HORI
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif

    Droite1
    DROITE  450  95   510 95  SOLID  1   0 0 0  SIMPLE
    TYPE_SENS_RESIZE =
```

```
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL =
    POSS_MOVE_RESIZE_GLOBAL =
    CONTRAINTE = 510 155
    MODIF   : FCT = FctModif

    Droite2
    DROITE  450  95 450  155  SOLID  1   0 0 0  SIMPLE
    TYPE_SENS_RESIZE =
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL =
    POSS_MOVE_RESIZE_GLOBAL =
    CONTRAINTE = 510 155
    MODIF   : FCT = FctModif

    Temps
    TEXTE  460 105  510 155 VARIABLE 50 CENTER  0 0 0 TRANSPARENT 0 0 0  ENTITE
    TYPE_SENS_RESIZE =
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL =
    POSS_MOVE_RESIZE_GLOBAL =
    CONTRAINTE = 510 155
    MODIF   : FCT = FctModif
  COMPOSANTS_ADMISSIBLES
 END

% Définition du noeud Mémoire

% Il s'agit d'un noeud dont le contour est assez complexe ( il est
% constitué de deux ellipses et de deux rectangles ). Le composant
% fixe est le nom de la mémoire. Ce noeud ne dispose pas de composant
% admissible.

NOEUD Memoire
 BEGIN
  CONTOUR
    TYPE_MOVE_COMP_ADM = ALL
    TYPE_RESIZE_COMP_ADM = ALL
    POSS_MOVE = UNLIMITED
    MODIF   : FCT = FctModif
    NAVIG   : FCT = FctNavig
    MOVE   : FCT = FctMove
    INFO : FCT = FctInfo
    DELETE : FCT = FctDelete
    INSERT : FCT = FctInsert
    ENCRAGE = 225 0 450 125 225 250 0 125
```

```
    ELLIPSE 0 190 450 250 SOLID  1   0 0 0  SOLID 255 255 255
    RECTANGLE 0 30 450 220 SOLID  1   0 0 0  SOLID 255 255 255
    ELLIPSE 0 0 450 60 SOLID  1   0 0 0  SOLID 255 255 255
    RECTANGLE 2 218 448 222 SOLID  1   255 255 255  SOLID 255 255 255
  COMPOSANTS_FIXES
    Nom
    TEXTE 10 70 440 180 VARIABLE  50 CENTER 0 0 0  OPAQUE 255 255 255 Texte
    TYPE_SENS_RESIZE =
    POSS_MOVE = NO
    POSS_MOVE_GLOBAL = VERTHORI
    TYPE_SENS_RESIZE_GLOBAL = VERTHORI
    POSS_MOVE_RESIZE_GLOBAL = VERTHORI
    CONTRAINTE = 0 0
    MODIF   : FCT = FctModif
  COMPOSANTS_ADMISSIBLES
 END

% Définition d'un arc

% Il s'agit d'un arc dont le contour est constitué d'une multi-droite.
% Cet arc ne possède pas d'élément fixe. Les noeuds sur lesquels peut
% porter l'arc sont tous les noeuds définis dans ce schéma (sauf
% bien entendu les unités organisationnellles ).

ARC  Arc
 BEGIN
  CONTOUR
    MODIF   : FCT = FctModif
    INFO : FCT = FctInfo
    DELETE : FCT = FctDelete
    INSERT : FCT = FctInsert
    MULTIDROITE 2  10 50   200 50  SOLID  1    0 0 0  FLECHED
  COMPOSANTS_FIXES
  NOEUD_ARC_ADMISSIBLES
    Memoire
    Message
    Message_Int
    Traitement
    Phase_Auto
    Phase_Manuelle
    Phase_Interactive
END
```

Annexe

Listing du programme ECLIPSE

Alain Van Hemelrijck

Eclipse.pas

```
{$x+}
program dlgbox;

{ Interface utilisateur utilisant Turbo Vision 6.0. Trois types d'entrées
  ont été définis : entier, réel, byte }


uses
  app,objects,menus,drivers,views,dialogs,msgbox,graphapp,graph,dos,
  stddlg,progr,introgr;

type

  { tmyapp : définition de l'application principale }

  pmyapp=^tmyapp;
  tmyapp=object(tapplication)
    constructor init;
  end;

  { tintegerinputline : définition d'une ligne d'entier }

  pintegerinputline=^tintegerinputline;
  tintegerinputline=object(tinputline)
    procedure handleevent(var event:tevent);virtual;
    function datasize:word;virtual;
    procedure getdata(var rec);virtual;
    procedure setdata(var rec);virtual;
    function valid(command:word):boolean;virtual;
  end;

  { tbyteinputline : définition d'une ligne byte }

  pbyteinputline=^tbyteinputline;
  tbyteinputline=object(tinputline)
    procedure handleevent(var event:tevent);virtual;
    function datasize:word;virtual;
    procedure getdata(var rec);virtual;
    procedure setdata(var rec);virtual;
    function valid(command:word):boolean;virtual;
  end;

  { trealinputline : définition d'une ligne réelle }

  prealinputline=^trealinputline;
  trealinputline=object(tinputline)
    procedure handleevent(var event:tevent);virtual;
    function datasize:word;virtual;
    procedure getdata(var rec);virtual;
    procedure setdata(var rec); virtual;
    function valid(command:word):boolean;virtual;
  end;

  { tmydialogbox : définition des boites de dialogues }

  pmydialogbox=^tmydialogbox;
  tmydialogbox=object(tdialog)
    constructor init(var bounds:trect;atitle:ttitlestr);
    procedure handleevent(var event:tevent);virtual;
    procedure save(mydata:data;orb:taborbite);
  end;

const
  cmmyok   =200;
```

```
  cmmyvue  =201;
  cmmyquit =cmcancel;
  cmmyload =203;
  cmmysave =204;

var
  myapp:tmyapp;
  orb:taborbite; { matrice de changement de repère des corps }

  procedure Egavgadriver;external; { driver écran : évite de charger le }
  {$L egavgadr}                    { gestionnaire écran à chaque fois }

  procedure runprg(mydata:data;orb:taborbite);    { lance la simulation }

  { variables d'entrées :
    mydata : données modifiables dans l'interface (voir définition des types)
    orb    : données non modifiables dans l'interface (matrice de changement
             de repère)                                                    }

  begin
   registerbgidriver(@Egavgadriver);
   if graphicsstart then
    begin
     progprinc(mydata,orb);
     graphicsstop;
    end;
   graphappdone;
  end;

 procedure runvue(var mydata:data;orb:taborbite); { lance l'interface visuelle }

  { variables d'entrées :
    mydata : données modifiables dans l'interface (voir définition des types)
    orb    : données non modifiables dans l'interface (matrice de changement
             de repère)                                                    }

 begin
  registerbgidriver(@Egavgadriver);
  if graphicsstart then begin
      obsvis(mydata,orb);
      graphicsstop;
  end;
  graphappdone;
 end;

 procedure charge(var orb:taborbite;var mydata:data;var charger:boolean);

 { chargement d'un fichier et mise des résultats dans les structures de
   données. Si aucun problème n'est apparu durant le chargement charger
   est vrai

   variables de sortie :
   mydata  : données modifiables dans l'interface (voir définition des types)
   orb     : données non modifiables dans l'interface (matrice de changement
             de repère)
   charger : précise si le chargement s'est effectué correctement
   }

  var
    dialog:pfiledialog;      { boîte de dialogue }
    r:trect;                 { les bornes de la boîte d'information }
    f:text;                  { le fichier }
    s:string;                { l'information sur le fichier }
    c:char;                  { variable auxiliaire }
```

```
  i,j,k:integer;          { variables auxiliaires }
  control1,control2:word; { contrôle des opérations de l'utilisateur }
  filename:pathstr;       { nom du fichier }

begin
  charger:=false;
  filename:='';

  { répéter le choix d'un fichier tant que l'utilisateur le désire }

  repeat
    dialog:=new(pfiledialog,init('*.ecl','select file','~f~ile name',fdokbutton,100));
    if dialog <> nil then begin
      control1:=desktop^.execview(dialog);
      if control1 <> cmcancel then begin
        dialog^.getfilename(filename);
        assign(f,filename);
        {$I-}
        reset(f);
        {$I+}
        charger:=ioresult=0;
        if not(charger) then exit;
        s:='';
        {$I-}
        readln(f,j);
        {$I+}
        charger:=ioresult=0;
        if not(charger) then exit;
        for i:=1 to j do
         begin
          {$I-}
          read(f,c);
          {$I+}
          charger:=ioresult=0;
          if not(charger) then exit;
          s:=s+c;
         end;
        {$I-}
        readln(f);
        {$I+}
        charger:=ioresult=0;
        if not(charger) then exit;
        r.assign(16,4,64,18);
        control2:=messageboxrect(r,s,nil,mfinformation+mfokbutton+mfcancelbutton);
        charger:= control2 = cmok;
      end;
      dispose(dialog,done);
    end;
  until (charger) or (control1=cmcancel);

  { chargement des paramètres }

  if charger then

   { chargement des matrices de changement de repère }

   begin
    for i:=1 to 3 do
     begin
      for j:=1 to 3 do
       for k:=1 to 3 do
        begin
         {$i-}
         readln(f,orb[i].repere[j,k]);
```

```
      {$i+}
      charger:=(ioresult=0) and not(eof(f));
      if not(charger) then exit;
     end;
   for j:=1 to 3 do
    begin
     {$i-}
     readln(f,orb[i].origine[j]);
     {$i+}
     charger:=(ioresult=0) and (not(eof(f)));
     if not(charger) then exit;
    end;
   {$I-}
   readln(f,k);
   {$I+}
   orb[i].dependant:=k=1;
   charger:=(ioresult=0) and (not(eof(f)));
   if not(charger) then exit;
   {$I-}
   readln(f,orb[i].corpdep);
   {$I+}
   if k=1 then charger:=(ioresult=0) and (not(eof(f))) and (orb[i].corpdep<4) and (orb[i].corpdep >0)
   else charger:=(ioresult=0) and (not(eof(f)));
   if not(charger) then exit;
 end;

{ chargement des valeurs modifiables par l'interface }

with mydata do
 begin

  { coordonnées observateur }

  {$i-}
  readln(f,d1);
  {$i+}
  charger:=(ioresult=0) and (not(eof(f)));
  if not(charger) then exit;
  {$i-}
  readln(f,d2);
  {$i+}
  charger:=(ioresult=0) and (not(eof(f)));
  if not(charger) then exit;
  {$i-}
  readln(f,d3);
  {$i+}
  charger:=(ioresult=0) and (not(eof(f)));
  if not(charger) then exit;

  { coordonnées point de visée }

  {$i-}
  readln(f,d4);
  {$i+}
  charger:=(ioresult=0) and (not(eof(f)));
  if not(charger) then exit;
  {$i-}
  readln(f,d5);
  {$i+}
  charger:=(ioresult=0) and (not(eof(f)));
  if not(charger) then exit;
  {$i-}
  readln(f,d6);
  {$i+}
```

```
charger:=(ioresult=0) and (not(eof(f)));
if not(charger) then exit;

{ angle de vision }

{$i-}
readln(f,d7);
{$i+}
charger:=(ioresult=0) and (not(eof(f)));
if not(charger) then exit;

{ définition du temps entre affichages successifs }

{$i-}
readln(f,d8);
{$i+}
charger:=(ioresult=0) and (not(eof(f)));
if not(charger) then exit;
{$i-}
readln(f,d9);
{$i+}
charger:=(ioresult=0) and (not(eof(f)));
if not(charger) then exit;

{ numéro du corps lumineux }

{$i-}
readln(f,d10);
{$i+}
charger:=(ioresult=0) and (not(eof(f))) and (d10>0) and (d10<4);
if not(charger) then exit;

{ le tableau de valeurs des corps }

{ corps C1 }

{$i-}
readln(f,d11);
{$i+}
charger:=(ioresult=0) and (not(eof(f)));
if not(charger) then exit;
{$i-}
readln(f,d12);
{$i+}
charger:=(ioresult=0) and (not(eof(f)));
if not(charger) then exit;
{$i-}
readln(f,d13);
{$i+}
charger:=(ioresult=0) and (not(eof(f)));
if not(charger) then exit;
{$i-}
readln(f,d14);
{$i+}
charger:=(ioresult=0) and (not(eof(f)));
if not(charger) then exit;
{$i-}
readln(f,d15);
{$i+}
charger:=(ioresult=0) and (not(eof(f)));
if not(charger) then exit;

{ corps C2 }
```

```
        {$i-}
        readln(f,d16);
        {$i+}
        charger:=(ioresult=0) and (not(eof(f)));
        if not(charger) then exit;
        {$i-}
        readln(f,d17);
        {$i+}
        charger:=(ioresult=0) and (not(eof(f)));
        if not(charger) then exit;
        {$i-}
        readln(f,d18);
        {$i+}
        charger:=(ioresult=0) and (not(eof(f)));
        if not(charger) then exit;
        readln(f,d19);
        {$i+}
        charger:=(ioresult=0) and (not(eof(f)));
        if not(charger) then exit;
        {$i-}
        readln(f,d20);
        {$i+}
        charger:=(ioresult=0) and (not(eof(f)));
        if not(charger) then exit;

        { corps C3 }

        {$i-}
        readln(f,d21);
        {$i+}
        charger:=(ioresult=0) and (not(eof(f)));
        if not(charger) then exit;
        readln(f,d22);
        {$i+}
        charger:=(ioresult=0) and (not(eof(f)));
        if not(charger) then exit;
        {$i-}
        readln(f,d23);
        {$i+}
        charger:=(ioresult=0) and (not(eof(f)));
        if not(charger) then exit;
        {$i-}
        readln(f,d24);
        {$i+}
        charger:=(ioresult=0) and (not(eof(f)));
        if not(charger) then exit;
        {$i-}
        readln(f,d25);
        {$i+}
        charger:=(ioresult=0);
        closefic(f);
      end;
    end;
  end;



{ tmydialogbox }

constructor tmydialogbox.init(var bounds:trect; atitle:ttitlestr);

{ initialisation de la fenêtre de dialogue à partir du titre et de ces bornes

  variables d'entrée :
```

```
  bounds : bornes de la fenêtre
  atitle : titre de la fenêtre }


var
  r:trect;              { coordonnées d'une zone }
  i:pintegerinputline;
begin
  tdialog.init(bounds,atitle);
  r.assign(2,2,32,3);
  insert(new(pstatictext,init(r,'Coordonnées de l''observateur <')));
  r.assign(33,2,44,3);
  i:=new(pintegerinputline,init(r,9));
  insert(i);
  r.assign(45,2,46,3);
  insert(new(pstatictext,init(r,',')));
  r.assign(47,2,58,3);
  insert(new(pintegerinputline,init(r,9)));
  r.assign(59,2,60,3);
  insert(new(pstatictext,init(r,',')));
  r.assign(61,2,72,3);
  insert(new(pintegerinputline,init(r,9)));
  r.assign(73,2,74,3);
  insert(new(pstatictext,init(r,'>')));
  r.assign(2,4,32,5);
  insert(new(pstatictext,init(r,'Coordonnées du pt de visée   <')));
  r.assign(33,4,44,5);
  insert(new(pintegerinputline,init(r,9)));
  r.assign(45,4,46,5);
  insert(new(pstatictext,init(r,',')));
  r.assign(47,4,58,5);
  insert(new(pintegerinputline,init(r,9)));
  r.assign(59,4,60,5);
  insert(new(pstatictext,init(r,',')));
  r.assign(61,4,72,5);
  insert(new(pintegerinputline,init(r,9)));
  r.assign(73,4,74,5);
  insert(new(pstatictext,init(r,'>')));
  r.assign(2,6,18,7);
  insert(new(pstatictext,init(r,'Angle de vision ')));
  r.assign(19,6,23,7);
  insert(new(pintegerinputline,init(r,2)));
  r.assign(2,8,30,9);
  insert(new(pstatictext,init(r,'Temps entre deux affichages ')));
  r.assign(31,8,37,9);
  insert(new(prealinputline,init(r,4)));
  r.assign(42,8,66,9);
  insert(new(pstatictext,init(r,'Incrémentation du temps ')));
  r.assign(67,8,73,9);
  insert(new(prealinputline,init(r,4)));
  r.assign(2,10,33,11);
  insert(new(pstatictext,init(r,'Le corps lumineux est le corps ')));
  r.assign(34,10,37,11);
  insert(new(pbyteinputline,init(r,1)));
  r.assign(11,12,78,13);
  insert(new(pstatictext,init(r,'Couleur  Rayon du corps  Rayon de l''orbite  Angle de départ  Année')));
  r.assign(2,13,10,14);
  insert(new(pstatictext,init(r,'Corps 1')));
  r.assign(12,13,16,14);
  insert(new(pintegerinputline,init(r,2)));
  r.assign(23,13,30,14);
  insert(new(pintegerinputline,init(r,5)));
  r.assign(40,13,48,14);
  insert(new(pintegerinputline,init(r,6)));
```

```
  r.assign(60,13,65,14);
  insert(new(pintegerinputline,init(r,3)));
  r.assign(72,13,78,14);
  insert(new(pintegerinputline,init(r,4)));
  r.assign(2,15,10,16);
  insert(new(pstatictext,init(r,'Corps 2')));
  r.assign(12,15,16,16);
  insert(new(pintegerinputline,init(r,2)));
  r.assign(23,15,30,16);
  insert(new(pintegerinputline,init(r,5)));
  r.assign(40,15,48,16);
  insert(new(pintegerinputline,init(r,6)));
  r.assign(60,15,65,16);
  insert(new(pintegerinputline,init(r,3)));
  r.assign(72,15,78,16);
  insert(new(pintegerinputline,init(r,4)));
  r.assign(2,17,10,18);
  insert(new(pstatictext,init(r,'Corps 3')));
  r.assign(12,17,16,18);
  insert(new(pintegerinputline,init(r,2)));
  r.assign(23,17,30,18);
  insert(new(pintegerinputline,init(r,5)));
  r.assign(40,17,48,18);
  insert(new(pintegerinputline,init(r,6)));
  r.assign(60,17,65,18);
  insert(new(pintegerinputline,init(r,3)));
  r.assign(72,17,78,18);
  insert(new(pintegerinputline,init(r,4)));
  r.assign(size.x div 2-35,20,size.x div 2-25,22);
  insert(new(pbutton,init(r,'S~t~art',cmmyok,bfnormal)));
  r.assign(size.x div 2-20,20,size.x div 2-10,22);
  insert(new(pbutton,init(r,'~V~ue',cmmyvue,bfnormal)));
  r.assign(size.x div 2-5,20,size.x div 2+6,22);
  insert(new(pbutton,init(r,'~C~harger',cmmyload,bfnormal)));
  r.assign(size.x div 2+10,20,size.x div 2+20,22);
  insert(new(pbutton,init(r,'~S~auver',cmmysave,bfnormal)));
  r.assign(size.x div 2+25,20,size.x div 2+36,22);
  insert(new(pbutton,init(r,'~Q~uitter',cmmyquit,bfnormal)));
  i^.select;
end;

procedure tmydialogbox.handleevent(var event:tevent);

{ gestion des événements de commandes

  variable d'entrée :
  event : dernier événement de l'interface }

var
  mydata:data;       { les données de l'interface }
  charger:boolean;   { résultat du chargement d'un fichier }

begin
  tdialog.handleevent(event);
  case event.what of
    evcommand:
      begin
        case event.command of

          { début de la simulation }

          cmmyok:if valid(cmok) then begin
                                       getdata(mydata);
                                       runprg(mydata,orb);
```

```
                                    end;

          { interface visuelle }

          cmmyvue:if valid(cmok) then begin
                                       getdata(mydata);
                                       runvue(mydata,orb);
                                       setdata(mydata);
                                      end;

          { chargement d'un fichier }

          cmmyload:begin
                     charge(orb,mydata,charger);
                     if charger then setdata(mydata)
                     else
                      begin
                       select;
                       write(#7);
                       messagebox(#3'Ce n''est pas un fichier acceptable',nil,mferror+mfokbutton);
                      end;
                    end;

          { sauvegarde des paramètres dans un fichier }

          cmmysave:begin
                     getdata(mydata);
                     save(mydata,orb);
                    end
        else
          exit;
        end;
        clearevent(event);
      end;
  end;
end;



procedure tmydialogbox.save(mydata:data;orb:taborbite);

{ sauvegarde des paramètres mydata et orb

  variables d'entrée :
   mydata  : données modifiables dans l'interface (voir définition des types)
   orb     : données non modifiables dans l'interface (matrice de changement
             de repère)}


var
  f:text;              { le fichier de sauvegarde }
  filename:pathstr;    { le nom du fichier de sauvegarde }
  info:string;         { l'information sur le fichier que l'on veut y sauver }
  dialog:pfiledialog;  { boîte de dialogue }
  i,j,k:integer;       { variables auxiliaires }

begin
  filename:='';
  info:='';

  { choix du nom de fichier de sauvegarde }

  if inputbox('information','',info,255)=cmok then begin
    dialog:=new(pfiledialog,init('*.ecl','enter file name','~f~ile name',fdokbutton,100));
      if dialog <> nil then begin
```

```
        if desktop^.execview(dialog) <> cmcancel then dialog^.getfilename(filename);
        dispose(dialog,done);
        assign(f,filename);
        rewrite(f);

        { sauvegarde de orb }

        writeln(f,length(info));
        writeln(f,info);
        for i:=1 to 3 do
         begin
          for j:=1 to 3 do
           for k:=1 to 3 do
             writeln(f,orb[i].repere[j,k]);
          for j:=1 to 3 do
            writeln(f,orb[i].origine[j]);
          if orb[i].dependant then
           writeln(f,'1')
          else
           writeln(f,'0');
          writeln(f,orb[i].corpdep);
         end;

        { sauvegarde de mydata }

        with mydata do
         begin
          writeln(f,d1);
          writeln(f,d2);
          writeln(f,d3);
          writeln(f,d4);
          writeln(f,d5);
          writeln(f,d6);
          writeln(f,d7);
          writeln(f,d8);
          writeln(f,d9);
          writeln(f,d10);
          writeln(f,d11);
          writeln(f,d12);
          writeln(f,d13);
          writeln(f,d14);
          writeln(f,d15);
          writeln(f,d16);
          writeln(f,d17);
          writeln(f,d18);
          writeln(f,d19);
          writeln(f,d20);
          writeln(f,d21);
          writeln(f,d22);
          writeln(f,d23);
          writeln(f,d24);
          writeln(f,d25);
          closefic(f);
         end;
       end;
  end;
end;

{ tintegerinputline }

procedure tintegerinputline.handleevent(var event:tevent);

{ vérification des valeurs acceptées pour des entiers

  variable d'entrée :
```

```
  event : dernier événement survenu

  variable de sortie :
  event : vide }

const
  activekey=[$0..$1f,$2b,$2d,$30..$39];

begin
  if ((event.what=evkeydown) and (event.keycode in activekey)) or (event.what <> evkeydown)
    then begin
      if (event.scancode=$1c) or (event.scancode=$f)
        then begin
          if valid(cmok) then owner^.selectnext(false);
          clearevent(event);
        end
        else tinputline.handleevent(event);
    end;
end;

function tintegerinputline.datasize:word;

{ taille de la ligne entière }

begin
  datasize:=sizeof(longint);
end;

procedure tintegerinputline.getdata(var rec);

{ mise de la ligne entière courante dans la variable rec

  variable de sortie :
  rec : contient l'entier }

var
  code:integer;
begin
  val(data^,longint(rec),code);
end;

procedure tintegerinputline.setdata(var rec);

{ initialisation d'une ligne entière

  variable d'entrée :
  rec : valeur à introduire dans la ligne entière courante }

begin
  str(longint(rec),data^);
  selectall(true);
end;

function tintegerinputline.valid(command:word):boolean;

{ validation d'un entier }

var
  code:integer;
  value:longint;
  ok:boolean;

begin
  ok:=true;
```

```
  if (command <> cmcancel) and (command <> cmvalid) then
  begin
    if data^='' then data^:='0';
    val(data^,value,code);
    if (code <> 0) then
    begin
      select;
      write(#7);
      messagebox(#3'Ce n''est pas une valeur entière correcte',nil,mferror+mfokbutton);selectall(true);
      ok:=false;
    end;
  end;
  if ok then valid:=tinputline.valid(command)
  else valid:=false;
end;

{ tbyteinputline }

procedure tbyteinputline.handleevent(var event:tevent);

{ vérification des valeurs acceptées pour des bytes
  variable d'entrée :
  event : dernier événement survenu

  variable de sortie :
  event : vide }

const
  activekey=[$0..$1f,$2b,$2d,$31..$33];

begin
  if ((event.what=evkeydown) and (event.keycode in activekey)) or (event.what <> evkeydown)
    then begin
      if (event.scancode=$1c) or (event.scancode=$f)
        then begin
          if valid(cmok) then owner^.selectnext(false);
          clearevent(event);
        end
        else tinputline.handleevent(event);
    end;
end;

function tbyteinputline.datasize:word;

begin
  datasize:=sizeof(byte);
end;

procedure tbyteinputline.getdata(var rec);

{ mise de la ligne byte courante dans la variable rec

  variable de sortie :
  rec : contient le byte }

var
  code:integer;

begin
  val(data^,byte(rec),code);
end;

procedure tbyteinputline.setdata(var rec);
```

```
{ Initialisation d'une ligne byte

  variable d'entrée :
  rec : valeur à introduire dans la ligne byte courante }

begin
  str(byte(rec),data^);
  selectall(true);
end;

function tbyteinputline.valid(command:word):boolean;

{ validation des bytes }

var
  code:integer;
  value:longint;
  ok:boolean;

begin
  ok:=true;
  if (command <> cmcancel) and (command <> cmvalid) then
  begin
    if data^='' then data^:='1';
    val(data^,value,code);
    if (code <> 0) then
    begin
      select;
      write(#7);
      messagebox(#3'Ce n''est pas una valeur de Byte correcte',nil,mferror+mfokbutton);selectall(true);
      ok:=false;
    end;
  end;
  if ok then valid:=tinputline.valid(command)
  else valid:=false;
end;

{ trealinputline }

procedure trealinputline.handleevent(var event:tevent);

{ vérification des valeurs acceptées pour des réels

  variable d'entrée :
  event : dernier événement survenu

  variable de sortie :
  event : vide }

const
  activekey=[$0..$1f,$2b,$2d,$2e,$30..$39];
begin
  if ((event.what=evkeydown) and (event.keycode in activekey)) or (event.what <> evkeydown)
    then begin
      if (event.scancode=$1c) or (event.scancode=$f)
        then begin
          if valid(cmok) then owner^.selectnext(false);
          clearevent(event);
        end
        else tinputline.handleevent(event);
    end;
end;

function trealinputline.datasize:word;
```

```
begin
  datasize:=sizeof(real);
end;

procedure trealinputline.getdata(var rec);

{ mise de la ligne réelle courante dans la variable rec

  variable de sortie :
  rec : contient le réel }

var
  code:integer;

begin
  val(data^,real(rec),code);
end;

procedure trealinputline.setdata(var rec);

{ initialisation d'une ligne réelle

  variable d'entrée :
  rec : valeur à introduire dans la ligne réelle courante }

begin
  if real(rec)<10 then
   str(real(rec):maxlen:2,data^)
  else
   if real(rec)<100 then
    str(real(rec):maxlen:1,data^)
   else
    str(real(rec):maxlen,data^);
  selectall(true);
end;

function trealinputline.valid(command:word):boolean;

{ validation des réels }

var
  code:integer;
  value:real;
  ok:boolean;
begin
  ok:=true;
  if (command <> cmcancel) and (command <> cmvalid) then
  begin
    if data^='' then data^:='0';
    val(data^,value,code);
    if (code <> 0) then
    begin
      select;
      write(#7);
      messagebox(#3'Ce n''est pas une valeur réelle correcte',nil,mferror+mfokbutton);selectall(true);
      ok:=false;
    end;
  end;
  if ok then valid:=tinputline.valid(command)
  else valid:=false;
end;
```

```
{ tmyapp }

constructor tmyapp.init;

{ initialisation de l'interface }

var
  dialog:pmydialogbox;
  r:trect;
  control:word;
  mydata:data;          { données modifiable dans l'interface }
  charger:boolean;


begin
  tapplication.init;
  r.assign(0,0,80,23);
  dialog:=new(pmydialogbox,init(r,'Simulateur d''éclipses'));
  repeat
   charge(orb,mydata,charger);
  until charger;
  dialog^.setdata(mydata);
  control:=desktop^.execview(dialog);
end;

begin
  myapp.init;
  myapp.run;
  myapp.done;
end.
```

Introgr.pas

```
unit introgr;

{ contient la procédure de l'interface visuelle }

interface

uses intro,perspect,calcul,progr,graph,crt,cosine;

procedure obsvis(var Mydata:data;orb:taborbite);

{ affichage les axes de C1 et la direction de vision. Permet de modifier
  les paramètres qui définissent la direction de vision

  variables d'entrée :
  mydata : données modifiables par l'interface principale
  orb : varaible non modifiables par l'interface

  variable de sortie :
  mydata : données modifiables par l'interface principale }

procedure closefic(var f:text);

{ fermeture d'un fichier texte : on a du la redéfinir car une fonction close
  existait dans turbo vision }

implementation

procedure closefic(var f:text);

begin
 close(f);
end;

procedure obsvis(var Mydata:data;orb:taborbite);

{ Affiche la direction de vision et les orbites. Elle permet de modifier la
  direction de vision }

var proj:projection;       { paramètre de la projection }
    persp:perspective;     { matrice de la perspective (changement de repère)}
    corps:tabcorp;         { contient les données de chacun des trois corps }
    mouv:tabpoint;         { contient les positions des corps sur leur orbite
                             dans le repère observateur }
    t:temps;               { variable auxiliaire }
    lum:byte;              { variable auxiliaire }
    obs,vis:vecteur;       { observateur et point de visée dans l'espace }
    max:longint;           { coordonnée maximum du point de visée et de l'observateur }
    pos:vecteur;           { position d'un point dans l'espace }
    xasp,yasp:word;        { longueur d'un pixel en x et en y }
    ratio:real;            { ratio de l'écran }
    fact:real;             { facteur de la projection en perspective }
    i,j:integer;
    xscreen,yscreen:integer; { coordonnée d'un point à l'écran }
    cx,cy:integer;         { centre de l'écran }
    sy: integer;           { demi longueur de l'écran en y }
    orx,ory:integer;       { origine d'une droite }
    obsx,obsy: integer;    { coordonnée de l'observateur à l'écran }
    visx,visy:integer;     { coordonnée du point de visée à l'écran }
    affiche,aux:string;
    a:char;
    plus:boolean;          { mode d'opération : Addition (true) soustraction (false) }
    page:byte;             { page de l'écran }
    stop:boolean;          { variable d'arrêt }
    vise:boolean;          { identification du point que l'on modifie :
```

```
                                true : point de visée
                                false : observateur }
    tau:real;
    tangammademi:real;     { tan(gamma/2) : défini le volume de vision }
    diffx,diffy,diffz:real; { différence par rapport au point de départ d'une position }

procedure calcul_xyecran(cx,cy:integer;fact,ratio:real;persp:perspective;pos:vecteur;var xscreen,yscreen:integer);

{ calcul les positions de pos (espace) à l'écran  et les met xscreen et yscreen

  variables d'entrée :
  cx, cy : centre de l'écran
  fact : facteur de la perspective
  ratio : ratio de l'écran
  persp : paramètre calculé de la projection en perspective
  pos : position du point dans l'espace
  xscreen, yscreen : position du point à l'écran }

var x,y,z:real;

begin

 { calcul de la position du point dans l'espace dans le repère de l'observateur }

 with persp do
  begin
   x:=-sinphi*(pos[1]-o[1])+cosphi*(pos[2]-o[2]);
   y:=-sintheta*(cosphi*(pos[1]-o[1])+sinphi*(pos[2]-o[2]))+costheta*(pos[3]-o[3]);
   z:=costheta*(cosphi*(pos[1]-o[1])+sinphi*(pos[2]-o[2]))-sintheta*o[3];
  end;

  { projection à l'écran }

 xscreen:=round(cx+x*fact/z*ratio);
 yscreen:=round(cy+y*fact/z);
end;

begin

 { initialisation }

 introdonnee(mydata,orb,t,lum,corps,proj);
 page:=0;
 plus:=true;
 vise:=false;
 for i:=1 to 3 do proj.v[i]:=0;
 proj.o[1]:=50000;
 proj.o[2]:=40000;
 proj.o[3]:=16666;
 proj.gamma:=20;
 perspecti(proj,persp);
 calcul_a_priori(corps,persp,mouv);
 setactivepage(page);
 setvisualpage(page);
 cx:=(getmaxx-60) div 2;
 cy:=getmaxy div 2;
 sy:=cy;
 getaspectratio(xasp,yasp);
 ratio:=yasp/xasp;
 tangammademi:=sinus(persp.gamma div 2)/cosinus(persp.gamma div 2);
 fact:=sy/tangammademi;
 with mydata do
  begin
   obs[1]:=d1;
```

```
  obs[2]:=d2;
  obs[3]:=d3;
  vis[1]:=d4;
  vis[2]:=d5;
  vis[3]:=d6;
 end;

{ répéter affichage tant que l'utilisateur ne veut pas sortir de l'interface
  individuelle }

repeat

 { affichage des commandes }

 setviewport(0,0,getmaxx,getmaxy,true);
 setcolor(14);
 outtextxy(20,0,'x');
 outtextxy(20,30,'y');
 outtextxy(20,60,'z');
 outtextxy(20,90,'X');
 outtextxy(20,120,'Y');
 outtextxy(20,150,'Z');
 outtextxy(20,300,'q');
 setcolor(15);
 line(60,0,60,350);
 outtextxy(0,315,'Quitter');
 if plus then
  begin
   outtextxy(0,10,'x=x+10');
   outtextxy(0,40,'y=y+10');
   outtextxy(0,70,'z=z+10');
   outtextxy(0,100,'x=x+100');
   outtextxy(0,130,'y=y+100');
   outtextxy(0,160,'z=z+100');
   outtextxy(0,190,'moins');
   setcolor(14);
   outtextxy(20,180,'-');
  end
 else
  begin
   outtextxy(0,10,'x=x-10');
   outtextxy(0,40,'y=y-10');
   outtextxy(0,70,'z=z-10');
   outtextxy(0,100,'x=x-100');
   outtextxy(0,130,'y=y-100');
   outtextxy(0,160,'z=z-100');
   outtextxy(0,190,'plus');
   setcolor(14);
   outtextxy(20,180,'+');
  end;
 if vise then
  begin
   setcolor(14);
   outtextxy(20,210,'o');
   setcolor(15);
   outtextxy(0,220,'Observa');
  end
 else
  begin
   setcolor(14);
   outtextxy(20,210,'v');
   setcolor(15);
   outtextxy(0,220,'Visée');
  end;
```

```
setviewport(60,0,getmaxx,getmaxy,true);

{ calcul de la projection adéquate }

repeat
 calcul_xyecran(cx,cy,fact,ratio,persp,vis,visx,visy);
 calcul_xyecran(cx,cy,fact,ratio,persp,obs,obsx,obsy);
  stop:=(obsx>2*cx) or (obsx<0) or (obsy > 2*cy) or (obsy<0) or
        (visx>2*cx) or (visx<0) or (visy > 2*cy) or (visy<0);
 if stop then
  begin
   if proj.gamma <90 then proj.gamma:=proj.gamma+4
   else
    begin
     for i:=1 to 3 do proj.o[i]:=proj.o[i]*2+1;
     proj.gamma:=12;
    end;
   perspecti(proj,persp);
   calcul_a_priori(corps,persp,mouv);
   tangammademi:=sinus(persp.gamma div 2)/cosinus(persp.gamma div 2);
   fact:=sy/tangammademi;
  end
 else
  begin
   stop:=(obsx>cx/2) and (obsx<3*cx/2) and (obsy >cy/2) and (obsy<3*cx/2)
        and (visx>cx/2) and (visx<3*cx/2) and (visy >cy/2) and (visy<3*cx/2);
   if stop then
    begin
     if proj.gamma >4 then proj.gamma:=proj.gamma-4
     else
      begin
       for i:=1 to 3 do proj.o[i]:=proj.o[i]div 2;
       proj.gamma:=12;
      end;
     perspecti(proj,persp);
     calcul_a_priori(corps,persp,mouv);
     tangammademi:=sinus(persp.gamma div 2)/cosinus(persp.gamma div 2);
     fact:=sy/tangammademi;
    end;
  end;
until not(stop);


{ affichages des axes }

setcolor(1);
for i:=1 to 3 do pos[i]:=0;
calcul_xyecran(cx,cy,fact,ratio,persp,pos,orx,ory);
max:=obs[1];
for i:=1 to 3 do
 if max<vis[i] then max:=vis[i];
for i:=2 to 3 do
 if max<obs[i] then max:=obs[i];
pos[1]:=max+20000;
pos[2]:=0;
pos[3]:=0;
calcul_xyecran(cx,cy,fact,ratio,persp,pos,xscreen,yscreen);
line(orx,ory,xscreen,yscreen);
outtextxy(5,247,'X');
pos[1]:=0;
pos[2]:=max+20000;
pos[3]:=0;
calcul_xyecran(cx,cy,fact,ratio,persp,pos,xscreen,yscreen);
line(orx,ory,xscreen,yscreen);
```

```
 begin
  if corps[i].dependant then
   begin
    tau:=corps[corps[i].corpdep].tau;
    diffx:=mouv[corps[i].corpdep,(round(tau)mod 360)div 10].x;
    diffx:=diffx+(mouv[corps[i].corpdep,(round(tau+10)mod 360)div 10].x-diffx)/10*(tau-round(tau)div 10 * 10);
    diffx:=diffx-mouv[corps[i].corpdep,0].x;
    diffy:=mouv[corps[i].corpdep,(round(tau)mod 360)div 10].y;
    diffy:=diffy+(mouv[corps[i].corpdep,(round(tau+10)mod 360)div 10].y-diffy)/10*(tau-round(tau)div 10 * 10);
    diffy:=diffy-mouv[corps[i].corpdep,0].y;
    diffz:=mouv[corps[i].corpdep,(round(tau)mod 360)div 10].z;
    diffz:=diffz+(mouv[corps[i].corpdep,(round(tau+10)mod 360)div 10].z-diffz)/10*(tau-round(tau)div 10 * 10);
    diffz:=diffz-mouv[corps[i].corpdep,0].z;
    with mouv[i,0] do
     begin
      xscreen:=round(cx+((x+diffx)*fact/(z+diffz))*ratio);
      yscreen:=round(cy+(y+diffy)*fact/(z+diffz));
      moveto(xscreen,yscreen);
     end;
    for j:=1 to 35 do
     with mouv[i,j] do
      begin
       xscreen:=round(cx+((x+diffx)*fact/(z+diffz))*ratio);
       yscreen:=round(cy+((y+diffy)*fact/(z+diffz)));
       lineto(xscreen,yscreen);
      end;
    with mouv[i,0] do
     begin
      xscreen:=round(cx+((x+diffx)*fact/(z+diffz))*ratio);
      yscreen:=round(cy+((y+diffy)*fact/(z+diffz)));
      lineto(xscreen,yscreen);
     end;
   end
  else
   begin
    with mouv[i,0] do
     begin
      xscreen:=round(cx+(x*fact/z)*ratio);
      yscreen:=round(cy+(y*fact/z));
      moveto(xscreen,yscreen);
     end;
    for j:= 1 to 35 do
     with mouv[i,j] do
      begin
       xscreen:=round(cx+(x*fact/z)*ratio);
       yscreen:=round(cy+(y*fact/z));
       lineto(xscreen,yscreen);
      end;
    with mouv[i,0] do
     begin
      xscreen:=round(cx+(x*fact/z)*ratio);
      yscreen:=round(cy+(y*fact/z));
      lineto(xscreen,yscreen);
     end;
   end;
  end;
setvisualpage(page);
a:=readkey;

{ gestion des demandes de l'utilisateur }

case a of

        { changement de la coordonnée x du point courant }
```

```
'x':if vise then
     begin
      if plus then vis[1]:=vis[1]+10
      else vis[1]:=vis[1]-10;
     end
    else
     if plus then obs[1]:=obs[1]+10
     else obs[1]:=obs[1]-10;

{ changement de la coordonnée x du point courant }

'X':if vise then
     begin
      if plus then vis[1]:=vis[1]+100
      else vis[1]:=vis[1]-100;
     end
    else
     if plus then obs[1]:=obs[1]+100
     else obs[1]:=obs[1]-100;

{ changement de la coordonnée y du point courant }

'y':if vise then
     begin
      if plus then vis[2]:=vis[2]+10
      else vis[2]:=vis[2]-10;
     end
    else
     if plus then obs[2]:=obs[2]+10
     else obs[2]:=obs[2]-10;

{ changement de la coordonnée y du point courant }

'Y':if vise then
     begin
      if plus then vis[2]:=vis[2]+100
      else vis[2]:=vis[2]-100;
     end
    else
     if plus then obs[2]:=obs[2]+100
     else obs[2]:=obs[2]-100;

{ changement de la coordonnée z du point courant }

'z':if vise then
     begin
      if plus then vis[3]:=vis[3]+10
      else vis[3]:=vis[3]-10;
     end
    else
     if plus then obs[3]:=obs[3]+10
     else obs[3]:=obs[3]-10;

{ changement de la coordonnée z du point courant }

'Z':if vise then
     begin
      if plus then vis[3]:=vis[3]+100
      else vis[3]:=vis[3]-100;
     end
    else
     if plus then obs[3]:=obs[3]+100
     else obs[3]:=obs[3]-100;
```

```
            { mise en mode addition }

            '+':plus:=true;

            { mise en mode soustraction }

            '-':plus:=false;

            { point courant = point de visée }

            'v':vise:=true;

            { point courant = position de l'observateur }

            'o':vise:=false;
  end;
  page:=(page+1) mod 2;
  setactivepage(page);
  cleardevice;
 until a='q';

 { sauvegarde des modifications }

 with mydata do
  begin
   d1:=obs[1];
   d2:=obs[2];
   d3:=obs[3];
   d4:=vis[1];
   d5:=vis[2];
   d6:=vis[3];
  end;
end;
end.
```

Progr.pas

```
unit progr;

{ contient le noyau de l'application }

interface

type

  { Données générales de la simulation }

  data = record
    d1,d2,d3,d4,d5,d6,d7:longint;
    d8,d9:real;
    d10:byte;
    d11,d12,d13,d14,d15,d16,d17,d18,d19,d20,d21,d22,d23,d24,d25:longint;
  end;

  matrice=array[1..3,1..3]of real;
  vecteur=array[1..3]of longint;

  { les matrices des orbites }

  orbite=record repere:matrice;
                origine:vecteur;
                corpdep:byte;
                dependant:boolean;
  end;
  taborbite=array[1..3]of orbite;

procedure progprinc (mydata:data;orb:taborbite);

{ variables d'entrée :
  mydata : données modifiables dans l'interface
  orb : données non modifiables dans l'interface }


implementation

uses intro,perspect,calcul,affichag;

procedure progprinc (mydata:data;orb:taborbite);

var corps : tabcorp;     { contient les données de chacun des trois corps }
    persp : perspective; { contient les paramètres de la perspective (costheta,sintheta,...)}
    proj  : projection;  { contient les données de la projection }
    mouv  : tabpoint;    { contient les positions des trois corps sur leur orbite
                           dans l'espace : mouv[1,1] contient la position par
                           rapport au repère observateur du corps pour un
                           angle de 10 de degrés sur son orbite }
    t:temps;             { temps entre deux affichages successifs }
    lum:byte;            { numéro du corps lumineux }
    a:char;

begin
 introdonnee(mydata,orb,t,lum,corps,proj);

 { calcul des paramètres de la perspective }

 perspecti(proj,persp);

 { calcul des mouvements des trois corps dans le repère observateur }

 calcul_a_priori(corps,persp,mouv);

 { calcul et affichage du mouvement des trois corps }
```

```
 affichage(t,lum,corps,mouv,persp);
end;
end.
```

Intro.pas

```
unit intro;

{ introduction des valeurs dans les structures du programme }

interface

uses crt,progr;

{ struture des paramètres d'un corps }

type corp = record r : longint;          { rayon de l'orbite }
                   rayon : longint;      { rayon du corps }
                   tau : real;           { angle de départ du corps sur son orbite }
                   dependant : boolean;  { dépendant d'un autre corps?}
                   corpdep : byte;       { numéro du corps dont il dépend }
                   repere : matrice;     { matrice de changement de repère vers C1 }
                   origine : vecteur;    { origine du repère dans le repère C1 }
                   an:integer;           { temps pour une révolution }
                   couleur:byte;         { couleur du corps }
            end;

      { structure des paramètres d'une projection }

      projection = record o : vecteur;     { position observateur }
                          v : vecteur;     { point de visée }
                          gamma : integer; { angle de vision }
                   end;
      tabcorp = array [1..3] of corp;
      temps = record t:real;
                     inc:real;
              end;

procedure introdonnee(mydata:data;orb:taborbite;var t:temps;var lum:byte;var corps:tabcorp;var proj:projection);

{ variables d'entrée :
  mydata : données modifiables par l'interface
  orb : données non modifiables par l'interface

  variables de sortie :
  t : temps entre deux affichages successifs et incrémentation de ce temps
  lum : numéro du corps lumineux
  corps : données sur les corps
  proj : paramètre de la projection en perspective }

implementation

procedure introdonnee(mydata:data;orb:taborbite;var t:temps;var lum:byte;var corps:tabcorp;var proj:projection);

var i,j,k,l:integer;

begin
 with mydata do
  begin
   with proj do
    begin
     o[1]:=d1;
     o[2]:=d2;
     o[3]:=d3;
     v[1]:=d4;
     v[2]:=d5;
     v[3]:=d6;
     gamma:=d7;
    end;
   t.t:=d8;
```

```
    t.inc:=d9;
    lum:=d10;
    with corps[1] do
     begin
      an:=d15;
      tau:=d14;
      r:=d13;
      rayon:=d12;
      couleur:=d11;
     end;
    with corps[2] do
     begin
      an:=d20;
      tau:=d19;
      r:=d18;
      rayon:=d17;
      couleur:=d16;
     end;
    with corps[3] do
     begin
      an:=d25;
      tau:=d24;
      r:=d23;
      rayon:=d22;
      couleur:=d21;
     end;
   end;
  for i:=1 to 3 do
   with orb[i] do
    begin
     for j:=1 to 3 do
      begin
       corps[i].origine[j]:=origine[j];
       for k:=1 to 3 do corps[i].repere[j,k]:=repere[j,k];
      end;
     corps[i].dependant:=dependant;
     corps[i].corpdep:=corpdep;
    end;
end;
end.
```

Perspect.pas

```
unit perspect;

{ contient la procédure qui calcule les paramètres de la perspective }

interface

uses intro,progr,cosine;

{ type définissant la matrice de chagement de repère du système vers
  l'observateur ainsi que l'angle de vision }

type perspective = record costheta : real;
                          sintheta : real;
                          cosphi : real;
                          sinphi : real;
                          gamma:integer;
                          o:vecteur;
                   end;

procedure perspecti(proj:projection;var persp:perspective);

{ procédure qui calcule les coéfficients de la matrice de transformation en
  perspective à partir des données contenues dans proj

  variable d'entrée :
  proj : paramètres de la projection en perspective

  variable de sortie :
  pers : paramètres calculés de la projection en perspective }


implementation

procedure perspecti(proj:projection;var persp:perspective);

var d : array[1..3] of real;     { direction de vision de l'observateur }
    distance : real; { distance entre l'observateur et le plan de projection }
    distproj : real; { projection sur le plan x,y de la distance entre
                       l'observateur et le plan de projection }
    i :integer;

begin
 inittab;
 persp.o:=proj.o;
 with proj do
  begin
   persp.gamma:=gamma;
   with persp do
    begin

     { calcul des angles du changement de repère }

     for i:=1 to 3 do d[i]:=o[i]-v[i];
     distance:=sqrt(d[1]*d[1]+d[2]*d[2]+d[3]*d[3]);
     distproj:=sqrt(d[1]*d[1]+d[2]*d[2]);
     costheta:=-distproj/distance;
     sintheta:=-d[3]/distance;
     if distproj<>0 then
      begin
       cosphi:=d[1]/distproj;
       sinphi:=d[2]/distproj;
      end
     else
      begin
```

```
        cosphi:=1.;
        sinphi:=0.;
       end;
     end;
   end;
end;
end.
```

Calcul.pas

```
unit calcul;

{ ensemble des calculs que l'on peut faire avant la simulation }

interface

uses intro,perspect,cosine;

{ représentaion d'un point dans l'espace grâce à ces coordonnées cartésiennes }

type point = record x:real;
                    y:real;
                    z:real;
             end;
     tabpoint = array [1..3,0..35] of point;

procedure calcul_a_priori(var corps:tabcorp;var persp:perspective;var mouv:tabpoint);

{ calcule pour chacun des trois corps de sa position sur une orbite tous les
  dix degrés. La position est données en coordonnées cartésiennes dans le
  repère de l'observateur

  variables d'entrée :
  corps : données sur les corps
  persp : paramètres calculés de la projection en perspective

  variable de sortie :
  mouv : ensemble des calculs des positions qui peuvent être fait avant la simulation }

implementation

procedure calcul_a_priori(var corps:tabcorp;var persp:perspective;var mouv:tabpoint);

var i,j:integer;
    pos:array[1..3] of real;      { position d'un corps d'un l'espace }
    ori:array[1..3] of real;      { translation de Ci vers observateur }
    mat:array[1..3,1..3] of real; { matrice de changement de repère (Ci vers
                                    observateur) }

begin

 { initialisation de la table des sinus et des cosinus }

 inittab;

 { calcul des positions }

 with persp do
  for i:=1 to 3 do
   with corps[i] do
    begin
     if dependant then
      begin
       origine[1]:=round(corps[corpdep].origine[1]+corps[corpdep].r*corps[corpdep].repere[1,1]);
       origine[2]:=round(corps[corpdep].origine[2]+corps[corpdep].r*corps[corpdep].repere[2,1]);
       origine[3]:=round(corps[corpdep].origine[3]+corps[corpdep].r*corps[corpdep].repere[3,1]);
      end;
     if an=0 then

      { calcul de la position des corps fixes }

      with mouv[i,0] do
       begin
        x:=-sinphi*(origine[1]-o[1])+cosphi*(origine[2]-o[2]);
```

```
        y:=-sintheta*(cosphi*(origine[1]-o[1])+sinphi*(origine[2]-o[2]))+costheta*(origine[3]-o[3]);
        z:=costheta*(cosphi*(origine[1]-o[1])+sinphi*(origine[2]-o[2]))-sintheta*o[3];
        for j:=1 to 35 do mouv[i,j]:=mouv[i,0];
       end
      else

       { calcul de la matrice de changement de repère du corps Ci vers
         l'observateur }

       begin
        for j:= 1 to 3 do
         begin
          mat[1,j]:=-sinphi*repere[1,j]+cosphi*repere[2,j];
          mat[2,j]:=-sintheta*cosphi*repere[1,j]-sintheta*sinphi*repere[2,j]+costheta*repere[3,j];
          mat[3,j]:=costheta*cosphi*repere[1,j]+costheta*sinphi*repere[2,j]+sintheta*repere[3,j];
         end;
        ori[1]:=-sinphi*(origine[1]-o[1])+cosphi*(origine[2]-o[2]);
        ori[2]:=-sintheta*cosphi*(origine[1]-o[1])-sintheta*sinphi*(origine[2]-o[2])+costheta*(origine[3]-o[3]);
        ori[3]:=costheta*cosphi*(origine[1]-o[1])+costheta*sinphi*(origine[2]-o[2])+sintheta*(origine[3]-o[3]);

        { calcul des positions d'orbites }

        for j:=0 to 35 do
         with mouv[i,j] do
          begin

           { calcul de la position du corps i pour un angle de j*10 degrés
             dans le repère associé au corps i. La coordonnée z est toujours
             égale à 0 puisque le corps tourne dans le plan XY de son repère }

           pos[1]:=r*cosinus(j*10);
           pos[2]:=r*sinus(j*10);


           { calcul de la position du corps i pour un angle de j*10 degrés
             dans le repère de l'observateur }

           x:=ori[1]+pos[1]*mat[1,1]+pos[2]*mat[1,2];
           y:=ori[2]+pos[1]*mat[2,1]+pos[2]*mat[2,2];
           z:=ori[3]+pos[1]*mat[3,1]+pos[2]*mat[3,2];

          end;
       end;
  end;
end;
end.
```

Affichag.pas

```
unit affichag;

{ Cette unit affiche et calcule la position des corps à l'écran }

interface

uses cosine,intro,perspect,calcul,graph,crt,tricone;

procedure affichage(t:temps;lum:byte;var corps:tabcorp;var mouv:tabpoint;var persp:perspective);

{ variables d'entrée :
  t : contient le temps entre deux affichages successifs et l'incrémentation
      de celui-ci
  lum : numéro du corps lumineux
  corps : données sur les corps (voir type corp)
  mouv : positions calculées au préalable des corps que l'on veut afficher
  persp : paramètre calculé de la projection en perspective }


implementation

procedure affichage(t:temps;lum:byte;var corps:tabcorp;var mouv:tabpoint;var persp:perspective);

var i,j:integer;
    temps:real;                   { temps écoulé dans la simulation }
    cx,cy:integer;                { coordonnée du centre de l'écran }
    sy:integer;                   { demi longueur de l'écran en y }
    centx:array[1..3] of integer; { coordonnée x de la projection du centre des corps }
    centy:array[1..3] of integer; { coordonnée y de la projection du centre des corps }
    rayon:array[1..3] of integer; { rayon projeté des corps }
    inittau:array[1..3] of real;  { angle de départ des corps sur leur orbite }
    tri:array[1..3] of byte;      { indice des corps suivant leur éloignement }
    diffx,diffy,diffz:real;       { différence des positions des corps par rapport à
                                    la position la plus proche inférieure dans mouv }
    fact:real;                    { facteur de la projection en perspective }
    centrex:array[1..3]of real;   { coordonnée x du centre des corps }
    centrey:array[1..3]of real;   { coordonnée x du centre des corps }
    centrez:array[1..3]of real;   { coordonnée x du centre des corps }
    page:word;                    { page écran }
    tau:real;                     { varaible auxiliaire contenant un angle de l'orbite }
    xasp,yasp:word;               { longueur et largeur d'un pixel }
    ratio:real;                   { ratio de l'écran }
    affiche:string;
    affcone:boolean;              { mode d'affichage : avec cône (True)
                                                       sans cône (False) }
    a:char;
    tangammademi:real;            { tan(gamma/2) }

begin

 { initialisation }

 inittab;
 affcone:=true;
 for i:=1 to 3 do inittau[i]:=corps[i].tau;
 temps:=0;
 page:=0;
 setactivepage(page);
 setvisualpage(page);
 getaspectratio(xasp,yasp);
 ratio:=yasp/xasp;
 cx:=(getmaxx-60) div 2;
 cy:=(getmaxy-50) div 2;
 sy:=cy;
```

```
tangammademi:=sinus(persp.gamma div 2)/cosinus(persp.gamma div 2);
fact:=sy/tangammademi;

{ calcul projection des corps fixes }

for i:=1 to 3 do
 with mouv[i,(round(corps[i].tau)mod 360)div 10] do
  begin
   if corps[i].an=0 then
    begin
     centx[i]:=round(cx+(x*fact/z)*ratio);
     centy[i]:=round(cy+(y*fact/z));
     rayon[i]:=round(corps[i].rayon*fact/z);
     centrex[i]:=x;
     centrey[i]:=y;
     centrez[i]:=z;
    end;
  end;

{ répéter l'affichage et les traitements des commandes tant que l'utilisateur
  ne désire pas sortir }

repeat

 { répéter l'affichage tant que l'utilisateur n'a pas frappé sur une touche }

 repeat
  temps:=temps+t.t;
  if temps<0 then temps:=360;
  page:=(page+1)mod 2;
  setactivepage(page);
  cleardevice;

  { affichage des commandes }

  setviewport(0,0,getmaxx,getmaxy,true);
  setcolor(15);
  outtextxy(0,10,'Zoom +');
  outtextxy(0,40,'Zoom -');
  outtextxy(0,70,'Pause');
  if affcone then
   outtextxy(0,100,'Ombre')
  else
   outtextxy(0,100,'Cône');
  outtextxy(0,130,'Temps');
  outtextxy(0,140,'  +');
  outtextxy(0,170,'Temps');
  outtextxy(0,180,'  -');
  outtextxy(0,210,'Temps');
  outtextxy(0,220,'  ++');
  outtextxy(0,250,'Temps');
  outtextxy(0,260,'  --');
  outtextxy(0,290,'Quitter');
  line(60,0,60,305);
  line(0,303,60,303);
  line(0,305,640,305);
  setcolor(14);
  outtextxy(20,0,'w');
  outtextxy(20,30,'x');
  outtextxy(20,60,'p');
  outtextxy(20,90,'c');
  outtextxy(20,120,chr(24));
  outtextxy(20,160,chr(25));
  affiche:='Shift '+chr(24);
```

```
outtextxy(0,200,affiche);
affiche:='Shift '+chr(25);
outtextxy(0,240,affiche);
outtextxy(20,280,'q');
str(temps:5:3,affiche);
affiche:='Temps : '+affiche+' jours';
outtextxy(0,325,affiche);
outtextxy(300,310,'Nombre de jours pour 1 révolution');
for i:=1 to 3 do
 begin
  setcolor(corps[i].couleur);
  if corps[i].an=0 then
   affiche:='Fixe'
  else
   str(corps[i].an,affiche);
  outtextxy(400,310+i*10,affiche);
  setcolor(corps[i].couleur);
 end;
setviewport(60,0,getmaxx,305,true);

{ calcul du temps courant }

for i:=1 to 3 do
 if corps[i].an<>0 then
  corps[i].tau:=temps/corps[i].an*360+inittau[i];

{ calcul des positions dans l'espace des corps en mouvement }

for i:=1 to 3 do
 if corps[i].an<>0 then
  with mouv[i,round(corps[i].tau)mod 360 div 10] do
   begin

    { interpolation linéaire }

    if corps[i].dependant then
     begin
      tau:=corps[corps[i].corpdep].tau;
      diffx:=mouv[corps[i].corpdep,(round(tau)mod 360)div 10].x;
      diffx:=diffx+(mouv[corps[i].corpdep,(round(tau+10)mod 360)div 10].x-diffx)/10*(tau-round(tau)div 10 * 10);
      diffx:=diffx-mouv[corps[i].corpdep,0].x;
      diffx:=diffx+(mouv[i,(round(corps[i].tau+10)mod 360)div 10].x-x)/10*(corps[i].tau-round(corps[i].tau)div 10 * 10);
      diffy:=mouv[corps[i].corpdep,(round(tau)mod 360)div 10].y;
      diffy:=diffy+(mouv[corps[i].corpdep,(round(tau+10)mod 360)div 10].y-diffy)/10*(tau-round(tau)div 10 * 10);
      diffy:=diffy-mouv[corps[i].corpdep,0].y;
      diffy:=diffy+(mouv[i,(round(corps[i].tau+10)mod 360)div 10].y-y)/10*(corps[i].tau-round(corps[i].tau)div 10 * 10);
      diffz:=mouv[corps[i].corpdep,(round(tau)mod 360)div 10].z;
      diffz:=diffz+(mouv[corps[i].corpdep,(round(tau+10)mod 360)div 10].z-diffz)/10*(tau-round(tau)div 10 * 10);
      diffz:=diffz-mouv[corps[i].corpdep,0].z;
      diffz:=diffz+(mouv[i,(round(corps[i].tau+10)mod 360)div 10].z-z)/10*(corps[i].tau-round(corps[i].tau)div 10 * 10);
     end
    else
     begin
      diffx:=(mouv[i,(round(corps[i].tau+10)mod 360)div 10].x-x)/10*(corps[i].tau-round(corps[i].tau)div 10 * 10);
      diffy:=(mouv[i,(round(corps[i].tau+10)mod 360)div 10].y-y)/10*(corps[i].tau-round(corps[i].tau)div 10 * 10);
      diffz:=(mouv[i,(round(corps[i].tau+10)mod 360)div 10].z-z)/10*(corps[i].tau-round(corps[i].tau)div 10 * 10);
     end;

    { calcul de la position dans l'espace }

    centrex[i]:=x+diffx;
    centrey[i]:=y+diffy;
    centrez[i]:=z+diffz;

    { calcul de la projection }
```

```
  centx[i]:=round(cx+(centrex[i]*fact/(centrez[i]))*ratio);
  centy[i]:=round(cy+(centrey[i]*fact/(centrez[i])));
  rayon[i]:=round(corps[i].rayon*fact/(centrez[i]));
 end;

{ tri des corps suivant leur éloignement }

if centrez[1]>=centrez[2] then
 begin
  if centrez[2]>=centrez[3] then
   begin
    tri[1]:=1;
    tri[2]:=2;
    tri[3]:=3;
   end
  else
   begin
    if centrez[1]>=centrez[3] then
     begin
      tri[1]:=1;
      tri[2]:=3;
      tri[3]:=2;
     end
    else
     begin
      tri[1]:=3;
      tri[2]:=1;
      tri[3]:=2;
     end;
   end;
 end
else
 begin
  if centrez[1]>=centrez[3] then
   begin
    tri[1]:=2;
    tri[2]:=1;
    tri[3]:=3;
   end
  else
   begin
    if centrez[2]>=centrez[3] then
     begin
      tri[1]:=2;
      tri[2]:=3;
      tri[3]:=1;
     end
    else
     begin
      tri[1]:=3;
      tri[2]:=2;
      tri[3]:=1;
     end;
   end;
 end;

for i:=1 to 3 do
 if tri[i]<>lum then
  begin

  { affichage des corps non lumineux }

  if (centrez[tri[i]]>0) and (rayon[tri[i]]>0) then
```

```
        extremite(centx[tri[i]],centy[tri[i]],centx[lum],centy[lum],rayon[lum],rayon[tri[i]],
                  centrex[lum],centrey[lum],centrez[lum],centrex[tri[i]],centrey[tri[i]],centrez[tri[i]],ratio,
                  affcone,corps[tri[i]].couleur);
      end
     else
      begin
       if centrez[lum]>0 then

        { affichage du corps lumineux }

        begin
         setfillstyle(1,corps[tri[i]].couleur);
         setcolor(corps[tri[i]].couleur);
         fillellipse(centx[tri[i]],centy[tri[i]],round(rayon[tri[i]]*ratio),rayon[tri[i]]);
        end;
      end;

  setvisualpage(page);
until keypressed;

{ gestion des commandes }

a:=readkey;
case a of #0:begin
             a:=readkey;

             { modification du temps entre deux affichages }

             case a of 'H':t.t:=t.t+t.inc;
                       'P':t.t:=t.t-t.inc;
                       'I':t.t:=t.t+t.inc*10;
                       'Q':t.t:=t.t-t.inc*10;
             end;
            end;

         { pause et affichage des orbites }

         'p':begin
             setcolor(4);
             for i:=1 to 3 do
              if corps[i].an<>0 then
               begin
                if corps[i].dependant then
                 begin
                  tau:=corps[corps[i].corpdep].tau;
                  diffx:=mouv[corps[i].corpdep,(round(tau)mod 360)div 10].x;
                  diffx:=diffx+(mouv[corps[i].corpdep,(round(tau+10)mod 360)div 10].x-diffx)/10*(tau-round(tau)div 10 * 10)
                  diffx:=diffx-mouv[corps[i].corpdep,0].x;
                  diffy:=mouv[corps[i].corpdep,(round(tau)mod 360)div 10].y;
                  diffy:=diffy+(mouv[corps[i].corpdep,(round(tau+10)mod 360)div 10].y-diffy)/10*(tau-round(tau)div 10 * 10)
                  diffy:=diffy-mouv[corps[i].corpdep,0].y;
                  diffz:=mouv[corps[i].corpdep,(round(tau)mod 360)div 10].z;
                  diffz:=diffz+(mouv[corps[i].corpdep,(round(tau+10)mod 360)div 10].z-diffz)/10*(tau-round(tau)div 10 * 10)
                  diffz:=diffz-mouv[corps[i].corpdep,0].z;
                  with mouv[i,0] do
                   begin
                    centx[i]:=round(cx+((x+diffx)*fact/(z+diffz))*ratio);
                    centy[i]:=round(cy+(y+diffy)*fact/(z+diffz));
                    moveto(centx[i],centy[i]);
                   end;
                  for j:=1 to 35 do
                   with mouv[i,j] do
                    begin
                     centx[i]:=round(cx+((x+diffx)*fact/(z+diffz))*ratio);
```

```
            centy[i]:=round(cy+((y+diffy)*fact/(z+diffz)));
            lineto(centx[i],centy[i]);
           end;
         with mouv[i,0] do
          begin
           centx[i]:=round(cx+((x+diffx)*fact/(z+diffz))*ratio);
           centy[i]:=round(cy+((y+diffy)*fact/(z+diffz)));
           lineto(centx[i],centy[i]);
          end;
        end
       else
        begin
         with mouv[i,0] do
          begin
           centx[i]:=round(cx+(x*fact/z)*ratio);
           centy[i]:=round(cy+(y*fact/z));
           moveto(centx[i],centy[i]);
          end;
         for j:= 1 to 35 do
          with mouv[i,j] do
           begin
            centx[i]:=round(cx+(x*fact/z)*ratio);
            centy[i]:=round(cy+(y*fact/z));
            lineto(centx[i],centy[i]);
           end;
         with mouv[i,0] do
          begin
           centx[i]:=round(cx+(x*fact/z)*ratio);
           centy[i]:=round(cy+(y*fact/z));
           lineto(centx[i],centy[i]);
          end;
        end;
       end;
    a:=readkey;
   end;

{ zoom avant }

'w':begin
     with persp do
      if gamma > 3 then
       begin
        gamma:=gamma div 2;
        tangammademi:=sinus(gamma div 2)/cosinus(gamma div 2);
        fact:=sy/tangammademi;
       end;
     for i:=1 to 3 do
      with mouv[i,(round(corps[i].tau)mod 360)div 10] do
       begin
        if corps[i].an=0 then
         begin
          centx[i]:=round(cx+(x*fact/z)*ratio);
          centy[i]:=round(cy+(y*fact/z));
          rayon[i]:=round(corps[i].rayon*fact/z);
          centrez[i]:=z;
         end;
       end;
    end;

{ zoom arrière }

'x':begin
     with persp do
      if gamma < 90 then
```

```
                begin
                 gamma:=gamma * 2;
                 tangammademi:=sinus(gamma div 2)/cosinus(gamma div 2);
                 fact:=sy/tangammademi;
                end;
               for i:=1 to 3 do
                with mouv[i,(round(corps[i].tau)mod 360)div 10] do
                 begin
                  if corps[i].an=0 then
                   begin
                    centx[i]:=round(cx+(x*fact/z)*ratio);
                    centy[i]:=round(cy+(y*fact/z));
                    rayon[i]:=round(corps[i].rayon*fact/z);
                    centrez[i]:=z;
                   end;
                 end;
              end;
          'c':affcone:=not(affcone);
  end;
 until (a='q');
end;
end.
```

Tricone.pas

```
unit tricone;

{ affichage d'un corps non lumineux et de son cône d'ombre }

interface
 uses graph,cosine;

 { structure des zones d'ombre }

 type ombre=array[1..56] of pointtype;
      triangle=array[1..3]of pointtype;

 procedure extremite(x,y,xp,yp,rayon1,rayon2:longint;xsol,ysol,zsol,xter,yter,zter,ratio:real;affcone:boolean;color:byte);

 { variables d'entrée :
   x, y : centre du corps non lumineux à l'écran
   xp, yp : centre du corps lumineux à l'écran
   rayon1 : rayon du corps lumineux à l'écran
   rayon2 : rayon du corps non lumineux à l'écran
   xsol, ysol, zsol : coordonnées du centre du corps lumineux dans le repère observateur
   xter, yter, zter : coordonnées du centre du corps non lumineux dans le repère observateur
   affcone : précise si il faut afficher le cône d'ombre
   color : couleur du corps non lumineux }

implementation

procedure extremite(x,y,xp,yp,rayon1,rayon2:longint;xsol,ysol,zsol,xter,yter,zter,ratio:real;affcone:boolean;color:byte);


var a,b,c,d:real;          { coefficient de l'équation de droite ax+b, cx+d }
    xfirst,xfirst2:boolean; { précise si la droite à une équation du type x=ay ou y=ax+b }
    sol1,sol2:real;
    dist,long:real;
    cone:triangle;          { la partie triangulaire du cône d'ombre }
    omb:ombre;              { la partie elliptique du cône }
    centx,centy:integer;
    rayon:real;
    alpha,beta,gam:integer;
    aux,aux1:real;
    aux2:integer;
    extx,exty:integer;
    i:integer;

procedure rechsol(x,y:longint;a,b,long:real;xfirst:boolean;var sol1,sol2:real);

{ recherche des deux points se trouvant à une distance long du point x,y sur
  la droite définie par a,b,xfirst

  variables d'entrée :
  x,y : coordonnée d'un point à l'écran
  a,b : paramètre de l'équation de la droite
  long : distance du point x,y
  xfisrt : type de l'équation de la droite:
             - true : x = ay
             - false : y= ax+b

  variables de sortie :
  sol1, sol2 : les deux solutions en x (xfirst = false) ou en y (xfisrt = true)          }

var aux:real;

begin
 if not(xfirst) then
  begin
```

```
    aux:=y-(c*(x-100)+d);
    sol1:=x+100*long/sqrt(10000+aux*aux);
    sol2:=x-100*long/sqrt(10000+aux*aux);
   end
  else
   begin
    sol1:=y+long;
    sol2:=y-long;
   end;
end;

procedure rechercheeq(x,y,xp,yp:longint;var a,b:real;var xfirst:boolean);

{ recherche l'équation de la droite passant par x,y et xp,yp

  variables d'entrée :
  x, y : coordonnée d'un point
  xp, yp : coordonnée d'un point

  variables de sortie :
  a, b : paramètre de l'équation de la droite
  xfisrt : type de l'équation de la droite:
             - true : x = ay
             - false : y= ax+b
  }

begin
 if x=xp then
  begin
   a:=0;
   b:=x;
   xfirst:=true;
  end
 else
  begin
   a:=(y-yp)/(x-xp);
   b:=y-a*x;
   xfirst:=false;
  end;
end;

function arctangente(a,b:real):word;

{ retourne arctan(-b/a) }

var angle:real;

begin
 if a=0 then
  begin
   if b>0 then arctangente:=90
   else arctangente:=270;
  end
 else
  begin
   angle:=arctan(-b/a)*180./pi;
   if a>0 then arctangente:=round(angle+180)
   else
    begin
     if angle<0 then arctangente:=round(angle+360)
     else arctangente:=round(angle);
    end;
  end;
end;
```

```
procedure droiteper(x,y:real;xfirst:boolean;var c,d:real;var xfirst2:boolean);

{ recherche l'équation de la droite perpendiculaire à celle définie
  par c,xfirst et passant par x,y

  variables d'entrée :
  x, y : coordonnée d'un point
  c : coéfficient angulaire de la droite dont on recherche la perpendiculaire
  xfirst : type de l'équation de la droite dont on recherche la perpendiculaire

  variables de sortie :
  c, d : paramètre de la droite perpendiculaire
  xfirst2 : type de l'équation de la droite perpendiculaire
  }

begin
 if c=0 then
  begin
   if xfirst then
    d:=y
   else
    d:=x;
   xfirst2:=not(xfirst);
  end
 else
  begin
   c:=-1/c;
   d:=-c*x+y;
   xfirst2:=false
  end;
end;

begin
 inittab;
 setcolor(8);
 setfillstyle(1,8);

 { droite passant par les deux centres }

 rechercheeq(x,y,xp,yp,a,b,xfirst);
 dist:=sqrt((x-xp)*(x-xp)+(y-yp)*(y-yp));
 if (dist<rayon2)  or (rayon1<=rayon2) then
  begin
   if zter>zsol then
    begin
     setcolor(color);
     setfillstyle(1,color);
    end;
   pieslice(x,y,0,360,round(rayon2*ratio))
  end
 else
  begin
   long:=rayon2*dist/(rayon1-rayon2);
   cone[1].x:=x+round((x-xp)/dist*long);
   cone[1].y:=y+round((y-yp)/dist*long);

   { recherche des deux autres extrémités du triangle }

   c:=a;
   droiteper(x,y,xfirst,c,d,xfirst2);
   rechsol(x,y,c,d,rayon2,xfirst2,sol1,sol2);
   if xfirst2 then
    begin
```

```
  cone[2].x:=x;
  cone[2].y:=round(sol1);
  cone[3].x:=x;
  cone[3].y:=round(sol2);
 end
else
 begin
  cone[2].x:=round(sol1);
  cone[2].y:=round(c*sol1+d);
  cone[3].x:=round(sol2);
  cone[3].y:=round(c*sol2+d);
 end;
if affcone then fillpoly(3,cone);
setcolor(color);
setfillstyle(1,color);
pieslice(x,y,0,360,round(rayon2*ratio));
setcolor(8);
setfillstyle(1,8);

{ calcul de la zone d'ombre }

long:=-rayon2*(zsol-zter)/sqrt((xsol-xter)*(xsol-xter)+(zsol-zter)*(zsol-zter)+(ysol-yter)*(ysol-yter));
extx:=x+round((x-xp)/dist*long);
exty:=y+round((y-yp)/dist*long);
rechercheeq(extx,exty,cone[2].x,cone[2].y,c,d,xfirst2);
aux:=extx+(cone[2].x-extx)/ 2.0;
aux1:=exty+(cone[2].y-exty)/ 2.0;
droiteper(aux,aux1,xfirst2,c,d,xfirst2);
if not(xfirst xor xfirst2) then
 begin
  if a<>c then
   begin
    centx:=round((b-d)/(c-a));
    centy:=round(a*(b-d)/(c-a)+b);
   end;
 end
else
 if xfirst then
  begin
   centx:=round(b);
   centy:=round(c*b+d);
  end
 else
  begin
   centx:=round(d);
   centy:=round(a*d+b);
  end;

{ ombre sur le corps }

aux2:=0;
alpha:=arctangente(x-cone[2].x,y-cone[2].y);
beta:=arctangente(x-cone[3].x,y-cone[3].y);
gam:=arctangente(x-cone[1].x,y-cone[1].y);
if alpha>beta then
 begin
  if (gam<alpha) and (gam>beta) then
   begin
    for i:=alpha div 10 downto beta div 10 do
     begin
      aux2:=aux2+1;
      omb[aux2].x:=round(x+rayon2*cosinus(i*10)*ratio);
      omb[aux2].y:=round(y-rayon2*sinus(i*10));
     end;
```

```
    end
   else
    for i:=alpha div 10 to (beta+360) div 10 do
     begin
      aux2:=aux2+1;
      omb[aux2].x:=round(x+rayon2*cosinus(i*10)*ratio);
      omb[aux2].y:=round(y-rayon2*sinus(i*10));
     end;
 end
else
 begin
  if (gam>alpha) and (gam<beta) then
   begin
    for i:=alpha div 10 to beta div 10 do
     begin
      aux2:=aux2+1;
      omb[aux2].x:=round(x+rayon2*cosinus(i*10)*ratio);
      omb[aux2].y:=round(y-rayon2*sinus(i*10));
     end;
   end
   else
    for i:=alpha div 10 downto (beta-360) div 10 do
     begin
      aux2:=aux2+1;
      omb[aux2].x:=round(x+rayon2*cosinus(i*10)*ratio);
      omb[aux2].y:=round(y-rayon2*sinus(i*10));
     end;
 end;
aux:=centx-extx;
aux1:=centy-exty;
rayon:=sqrt(aux*aux+aux1*aux1);
if (a<>c) and (rayon<5*rayon2) then
 begin
  alpha:=arctangente(centx-cone[2].x,centy-cone[2].y);
  beta:=arctangente(centx-cone[3].x,centy-cone[3].y);
  gam:=arctangente(aux,aux1);
  if alpha>beta then
   begin
    if (alpha>gam) and (beta<gam) then
     for i:=beta div 10 to alpha div 10 do
      begin
       aux2:=aux2+1;
       omb[aux2].x:=round(centx+rayon*cosinus(i*10));
       omb[aux2].y:=round(centy-rayon*sinus(i*10));
      end
    else
     for i:=beta div 10 downto (alpha-360) div 10 do
      begin
       aux2:=aux2+1;
       omb[aux2].x:=round(centx+rayon*cosinus(i*10));
       omb[aux2].y:=round(centy-rayon*sinus(i*10));
      end;
   end
  else
   begin
    if (alpha<gam) and (beta>gam) then
     for i:=beta div 10 downto alpha div 10 do
      begin
       aux2:=aux2+1;
       omb[aux2].x:=round(centx+rayon*cosinus(i*10));
       omb[aux2].y:=round(centy-rayon*sinus(i*10));
      end
    else
     for i:=beta div 10 to (alpha+360) div 10 do
```

```
        begin
         aux2:=aux2+1;
         omb[aux2].x:=round(centx+rayon*cosinus(i*10));
         omb[aux2].y:=round(centy-rayon*sinus(i*10));
        end;
      end;
    end
   else
    begin
     aux2:=aux2+1;
     omb[aux2]:=omb[1];
    end;
   for i:=aux2+1 to 56 do omb[i]:=omb[aux2];
   fillpoly(56,omb);
  end;
end;
end.
```

Cosine.pas

```
unit cosine;

{ Cette unit contient les procédures qui permettent de calculer les sinus et
  cosinus d'une variable entière }

interface
 var tab:array[0..90]of real;
 procedure inittab;
 function cosinus(i:integer):real;
 function sinus(i:integer):real;

implementation

 procedure inittab;

 begin
  tab[0]:= 1.0;
  tab[1]:= 9.9984769515E-01;
  tab[2]:= 9.9939082701E-01;
  tab[3]:= 9.9862953475E-01;
  tab[4]:= 9.9756405026E-01;
  tab[5]:= 9.9619469809E-01;
  tab[6]:= 9.9452189537E-01;
  tab[7]:= 9.9254615164E-01;
  tab[8]:= 9.9026806874E-01;
  tab[9]:= 9.8768834059E-01;
  tab[10]:= 9.8480775301E-01;
  tab[11]:= 9.8162718345E-01;
  tab[12]:= 9.7814760073E-01;
  tab[13]:= 9.7437006478E-01;
  tab[14]:= 9.7029572628E-01;
  tab[15]:= 9.6592582629E-01;
  tab[16]:= 9.6126169594E-01;
  tab[17]:= 9.5630475596E-01;
  tab[18]:= 9.5105651629E-01;
  tab[19]:= 9.4551857560E-01;
  tab[20]:= 9.3969262079E-01;
  tab[21]:= 9.3358042650E-01;
  tab[22]:= 9.2718385457E-01;
  tab[23]:= 9.2050485345E-01;
  tab[24]:= 9.1354545764E-01;
  tab[25]:= 9.0630778704E-01;
  tab[26]:= 8.9879404630E-01;
  tab[27]:= 8.9100652419E-01;
  tab[28]:= 8.8294759286E-01;
  tab[29]:= 8.7461970714E-01;
  tab[30]:= 8.6602540378E-01;
  tab[31]:= 8.5716730070E-01;
  tab[32]:= 8.4804809616E-01;
  tab[33]:= 8.3867056795E-01;
  tab[34]:= 8.2903757255E-01;
  tab[35]:= 8.1915204429E-01;
  tab[36]:= 8.0901699438E-01;
  tab[37]:= 7.9863551005E-01;
  tab[38]:= 7.8801075361E-01;
  tab[39]:= 7.7714596146E-01;
  tab[40]:= 7.6604444312E-01;
  tab[41]:= 7.5470958022E-01;
  tab[42]:= 7.4314482548E-01;
  tab[43]:= 7.3135370162E-01;
  tab[44]:= 7.1933980034E-01;
  tab[45]:= 7.0710678119E-01;
  tab[46]:= 6.9465837046E-01;
  tab[47]:= 6.8199836006E-01;
```

```
 tab[48]:= 6.6913060636E-01;
 tab[49]:= 6.5605902899E-01;
 tab[50]:= 6.4278760969E-01;
 tab[51]:= 6.2932039105E-01;
 tab[52]:= 6.1566147533E-01;
 tab[53]:= 6.0181502315E-01;
 tab[54]:= 5.8778525229E-01;
 tab[55]:= 5.7357643635E-01;
 tab[56]:= 5.5919290347E-01;
 tab[57]:= 5.4463903502E-01;
 tab[58]:= 5.2991926423E-01;
 tab[59]:= 5.1503807491E-01;
 tab[60]:= 5.0000000000E-01;
 tab[61]:= 4.8480962025E-01;
 tab[62]:= 4.6947156279E-01;
 tab[63]:= 4.5399049974E-01;
 tab[64]:= 4.3837114679E-01;
 tab[65]:= 4.2261826174E-01;
 tab[66]:= 4.0673664308E-01;
 tab[67]:= 3.9073112849E-01;
 tab[68]:= 3.7460659342E-01;
 tab[69]:= 3.5836794955E-01;
 tab[70]:= 3.4202014333E-01;
 tab[71]:= 3.2556815446E-01;
 tab[72]:= 3.0901699438E-01;
 tab[73]:= 2.9237170472E-01;
 tab[74]:= 2.7563735582E-01;
 tab[75]:= 2.5881904510E-01;
 tab[76]:= 2.4192189560E-01;
 tab[77]:= 2.2495105434E-01;
 tab[78]:= 2.0791169082E-01;
 tab[79]:= 1.9080899538E-01;
 tab[80]:= 1.7364817767E-01;
 tab[81]:= 1.5643446504E-01;
 tab[82]:= 1.3917310096E-01;
 tab[83]:= 1.2186934340E-01;
 tab[84]:= 1.0452846327E-01;
 tab[85]:= 8.7155742748E-02;
 tab[86]:= 6.9756473744E-02;
 tab[87]:= 5.2335956243E-02;
 tab[88]:= 3.4899496703E-02;
 tab[89]:= 1.7452406437E-02;
 tab[90]:= 0.0;

end;

function cosinus(i:integer):real;

begin
 i:=i mod 360;
 if i<0 then i:=i+360;
 if i<=90
 then
    cosinus:=tab[i]
 else
    if i<=180
    then
       cosinus:=-tab[180-i]
    else
       if i<=270
       then
          cosinus:=-tab[i-180]
       else
          cosinus:=tab[360-i];
```

```
end;

function sinus(i:integer):real;

begin
 i:=i mod 360;
 if i<0 then i:=i+360;
 if i<=90
 then
    sinus:=tab[90-i]
 else
    if i<=180
    then
       sinus:=tab[i-90]
    else
       if i<=270
       then
          sinus:=-tab[270-i]
       else
          sinus:=-tab[i-270];
 end;
end.
```