Un système d'interprétation graphique de données : application à l'illustration dynamique de programmes

Christian Laugier

▶ **To cite this version:**

Christian Laugier. Un système d'interprétation graphique de données : application à l'illustration dynamique de programmes. Modélisation et simulation. Institut National Polytechnique de Grenoble - INPG; Université Joseph-Fourier - Grenoble I, 1976. Français. NNT : . tel-00287050

HAL Id: tel-00287050

https://theses.hal.science/tel-00287050

Submitted on 10 Jun 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE

présentée à

Université Scientifique et Médicale de Grenoble

Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR DE 3ème CYCLE

«Informatique»

par

Christian LAUGIER

UN SYSTEME D'INTERPRETATION GRAPHIQUE DE DONNEES. APPLICATION A L'ILLUSTRATION DYNAMIQUE DE PROGRAMMES.

Thèse soutenue le 28 octobre 1976 devant la Commission d'Examen :

Président : J. KUNTZMANN

Examinateurs : G. VEILLON
Ph. JORRAND
M. LUCAS

UNIVERSITE SCIENTIFIQUE
ET MEDICALE DE GRENOBLE

Monsieur Gabriel CAU : Président
Monsieur Pierre JULLIEN : Vice-Président

MEMBRES DU CORPS ENSEIGNANT DE L'U.S.M.G.

PROFESSEURS TITULAIRES

MM.	ARNAUD Paul	Chimie
	AUBERT Guy	Physique
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique Expérimentale
	BARBIER Reynold	Géologie appliquée
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale
	BEAUDOING André	Clinique de Pédiatrie et Puériculture
	BERNARD Alain	Mathématiques Pures
Mme	BERTRANDIAS Françoise	Mathématiques Pures
MM.	BERTRANDIAS Jean-Paul	Mathématiques Pures
	BEZES Henri	Pathologie chirurgicale
	BLAMBERT Maurice	Mathématiques Pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Georges	Electrotechnique
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET-EYMARD Joseph	Clinique gastro-entérologique
Mme	BONNIER Marie-Jeanne	Chimie générale
MM.	BOUCHERLE André	Chimie et toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BOUSSARD Jean-Claude	Mathématiques Appliquées
	BOUTET DE MONTVEL Louis	Mathématiques Pures
	BRAVARD Yves	Géographie
	CABANEL Guy	Clinique rhumatologique et hydrologique
	CALAS François	Anatomie
	CARLIER Georges	Biologie végétale
	CARRAZ Gilbert	Biologie animale et pharmacodynamie
	CAU Gabriel	Médecine légale et toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques Pures
	CHARACHON Robert	Clinique Oto-rhino-laryngologique
	CHATEAU Robert	Clinique de neurologie
	CHIBON Pierre	Biologie animale
	COEUR André	Pharmacie chimique et chimie analytique
	CONTAMIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie pathologique
Mme	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBELMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DELORMAS Pierre	Pneumophtisiologie

MM.	DEPORTES Charles	Chimie minérale
	DESRE Pierre	Métallurgie
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée (IUT A)
	DOLIQUE Jean-Michel	Physique des plasmas
	DREYFUS Bernard	Thermodynamique
	DUCROS Pierre	Cristallographie
	DUGOIS Pierre	Clinique de dermatologie et syphiligraphie
	GAGNAIRE Didier	Chimie physique
	GALLISSOT François	Mathématiques Pures
	GALVANI Octave	Mathématiques Pures
	GASTINEL Noël	Analyse numérique
	GAVEND Michel	Pharmacologie
	GEINDRE Michel	Electroradiologie
	GERBER Robert	Mathématiques Pures
	GERMAIN Jean-Pierre	Mécanique
	GIRAUD Pierre	Géologie
	JANIN Bernard	Géographie
	KAHANE André	Physique générale
	KLEIN Joseph	Mathématiques Pures
	KOSZUL Jean-Louis	Mathématiques Pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques Appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
Mme	LAJZEROWICZ Janine	Physique
MM.	LAJZEROWICZ Joseph	Physique
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie pharmaceutique
	LAURENT Pierre-Jean	Mathématiques Appliquées
	LEDRU Jean	Clinique médicale B
	LLIBOUTRY Louis	Géophysique
	LOISEAUX Pierre	Sciences nucléaires
	LONGEQUEUE Jean-Pierre	Physique nucléaire
	LOUP Jean	Géographie
Mlle	LUTZ Elisabeth	Mathématiques Pures
MM.	MALGRANGE Bernard	Mathématiques Pures
	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Clinique cardiologique
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et Pétrographie
	MICOUD Max	Clinique maladies infectieuses
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	MULLER Jean-Michel	Thérapeutique (Néphrologie)
	NEEL Louis	Physique du Solide
	OZENDA Paul	Botanique
	PAYAN Jean-Jacques	Mathématiques Pures
	PEBAY-PEYROULA Jean-Claude	Physique
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	REVOL Michel	Urologie
	RINALDI Renaud	Physique
	DE ROUGEMONT Jacques	Neuro-chirurgie
	SEIGNEURIN Raymond	Microbiologie et Hygiène
	SENGEL Philippe	Zoologie

MM. SIBILLE Robert	Construction mécanique (IUT A)
SOUTIF Michel	Physique générale
TANCHE Maurice	Physiologie
TRAYNARD Philippe	Chimie générale
VAILLANT François	Zoologie
VALENTIN Jacques	Physique nucléaire
VAUQUOIS Bernard	Calcul électronique
Mme VERAIN Alice	Pharmacie galénique
MM. VERAIN André	Physique
VEYRET Paul	Géographie
VIGNAIS Pierre	Biochimie médicale
YOCCOZ Jean	Physique nucléaire théorique

PROFESSEURS ASSOCIES

MM. CLARK Gilbert	Spectrométrie physique
CRABBE Pierre	CERMO
ENGLMAN Robert	Spectrométrie physique
HOLTZBERG Frédéric	Basses températures
DEMBICKI Eugéniuz	Mécanique
MATSUSHIMA Yozo	Mathématiques Pures

PROFESSEURS SANS CHAIRE

Mlle AGNIUS-DELORD Claudine	Physique pharmaceutique
ALARY Josette	Chimie analytique
MM. AMBROISE-THOMAS Pierre	Parasitologie
BELORIZKY Elie	Physique
BENZAKEN Claude	Mathématiques Appliquées
BIAREZ Jean-Pierre	Mécanique
BILLET Jean	Géographie
BOUCHET Yves	Anatomie
BRUGEL Lucien	Energétique (IUT A)
BUISSON René	Physique (IUT A)
BUTEL Jean	Orthopédie
COHEN ADDAD Pierre	Spectrométrie physique
COLOMB Maurice	Biochimie
CONTE René	Physique (IUT A)
DEPASSEL Roger	Mécanique des fluides
FONTAINE Jean-Marc	Mathématiques Pures
GAUTHIER Yves	Sciences Biologiques
GAUTRON René	Chimie
GIDON Paul	Géologie et Minéralogie
GLENAT René	Chimie organique
GROULADE Joseph	Biochimie médicale
HACQUES Gérard	Calcul numérique
HOLLARD Daniel	Hématologie
HUGONOT Robert	Hygiène et Médecine préventive
IDELMAN Simon	Physiologie animale
JOLY Jean-René	Mathématiques Pures
JULLIEN Pierre	Mathématiques Appliquées
Mme KAHANE Josette	Physique
MM. KRAKOWIAK Sacha	Mathématiques Appliquées
KUHN Gérard	Physique (IUT A)
LE ROY Philippe	Mécanique (IUT A)
LUU DUC Cuong	Chimie organique

MM.	MAYNARD Roger	Physique du solide
Mme	MINIER Colette	Physique (IUT A)
MM.	PELMONT Jean	Biochimie
	PERRIAUX Jean-Jacques	Géologie et Minéralogie
	PFISTER Jean-Claude	Physique du solide
Mlle	PIERY Yvette	Physiologie animale
MM.	RAYNAUD Hervé	M.I.A.G.
	REBECQ Jacques	Biologie (CUS)
	REYMOND Jean-Charles	Chirurgie générale
	RICHARD Lucien	Biologie végétale
Mme	RINAUDO Marguerite	Chimie macromoléculaire
MM.	ROBERT André	Chimie papetière
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIROT Louis	Chirurgie générale
Mme	SOUTIF Jeanne	Physique générale
MM.	STREGLITZ Paul	Anesthésiologie
	VIALON Pierre	Géologie
	VAN CUTSEM Bernard	Mathématiques Appliquées

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

MM.	AMBLARD Pierre	Dermatologie
	ARMAND Gilbert	Géographie
	ARMAND Yves	Chimie (IUT A)
	BACHELOT Yvan	Endocrinologie
	BARGE Michel	Neuro-chirurgie
	BARJOLLE Michel	M.I.A.G.
	BEGUIN Claude	Chimie organique
Mme	BERIEL Hélène	Pharmacodynamie
MM.	BOST Michel	Pédiatrie
	BOUCHARLAT Jacques	Psychiatrie adultes
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BRODEAU François	Mathématiques (IUT B)
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse
	CHARDON Michel	Géographie
	CHERADAME Hervé	Chimie papetière
	CHIAVERINA Jean	Biologie appliquée (EFP)
	CONTAMIN Charles	Chirurgie thoracique et cardio-vasculaire
	CORDONNIER Daniel	Néphrologie
	COULOMB Max	Radiologie
	CROUZET Guy	Radiologie
	CYROT Michel	Physique du solide
	DELOBEL Claude	M.I.A.G.
	DENIS Bernard	Cardiologie
	DOUCE Roland	Physiologie végétale
	DUSSAUD René	Mathématiques (CUS)
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	FAURE Gilbert	Urologie
	GAUTIER Robert	Chirurgie générale
	GENSAC Pierre	Botanique
	GIDON Maurice	Géologie
	GROS Yves	Physique (IUT A)

MM. GUITTON Jacques	Chimie
HICTER Pierre	Chimie
IVANES Marcel	Electricité
JALBERT Pierre	Histologie
JUNIEN-LAVILLAVROY Claude	O.R.L.
KOLODIE Lucien	Hématologie
LE NOC Pierre	Bactériologie-virologie
LEROY Philippe	IUT A
MACHE Régis	Physiologie végétale
MAGNIN Robert	Hygiène et médecine préventive
MALLION Jean-Michel	Médecine du travail
MARECHAL Jean	Mécanique (IUT A)
MARTIN-BOUYER Michel	Chimie (CUS)
MICHOULIER Jean	Physique (IUT A)
NEGRE Robert	Mécanique (IUT A)
NEMOZ Alain	Thermodynamique
NOUGARET Marcel	Automatique (IUT A)
PARAMELLE Bernard	Pneumologie
PECCOUD François	Analyse (IUT B)
PEFFEN René	Métallurgie (IUT A)
PERRET Jean	Neurologie
PERRIER Guy	Géophysique - Glaciologie
PHELIP Xavier	Rhumatologie
RACHAIL Michel	Médecine interne
RACINET Claude	Gynécologie et obstétrique
RAMBAUD André	Hygiène et hydrologie
RAMBAUD Pierre	Pédiatrie
Mme RENAUDET Jacqueline	Bactériologie
MM. ROBERT Jean-Bernard	Chimie Physique
ROMIER Guy	Mathématiques (IUT B)
SHOM Jean-Claude	Chimie générale
STOEBNER Pierre	Anatomie pathologique
VROUSOS Constantin	Radiologie

MAITRE DE CONFERENCES ASSOCIES

M. COLE Antony	Sciences nucléaires

Fait à SAINT MARTIN D'HERES, AVRIL 1976.

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : M. Philippe TRAYNARD

Vice-Président : M. Pierre-Jean LAURENT

PROFESSEURS TITULAIRES

MM. BENOIT Jean	Radioélectricité
BESSON Jean	Electrochimie
BLOCH Daniel	Physique du solide
BONNETAIN Lucien	Chimie Minérale
BONNIER Etienne	Electrochimie et Electrométallurgie
BOUDOURIS Georges	Radioélectricité
BRISSONNEAU Pierre	Physique du solide
BUYLE-BODIN Maurice	Electronique
COUMES André	Radioélectricité
DURAND Francis	Métallurgie
FELICI Noël	Electrostatique
FOULARD Claude	Automatique
LESPINARD Georges	Mécanique
MOREAU René	Mécanique
PARIAUD Jean-Charles	Chimie-Physique
PAUTHENET René	Physique du solide
PERRET René	Servomécanismes
POLOUJADOFF Michel	Electrotechnique
SILBER Robert	Mécanique des Fluides

PROFESSEUR ASSOCIE

M. ROUXEL Roland	Automatique

PROFESSEURS SANS CHAIRE

MM. BLIMAN Samuel	Electronique
BOUVARD Maurice	Génie Mécanique
COHEN Joseph	Electrotechnique
LACOUME Jean-Louis	Géophysique
LANCIA Roland	Electronique
ROBERT François	Analyse numérique
VEILLON Gérard	Informatique Fondamentale et Appliquée
ZADWORNY François	Electronique

MAITRES DE CONFERENCES

MM. ANCEAU François	Mathématiques Appliquées
CHARTIER Germain	Electronique
GUYOT Pierre	Chimie Minérale
IVANES Marcel	Electrotechnique
JOUBERT Jean-Claude	Physique du solide
MORET Roger	Electrotechnique Nucléaire
PIERRARD Jean-Marie	Mécanique
SABONNADIERE Jean-Claude	Informatique Fondamentale et Appliquée
Mme SAUCIER Gabrièle	Informatique Fondamentale et Appliquée

MAITRE DE CONFERENCES ASSOCIE

M. LANDAU Ioan	Automatique

CHERCHEURS DU C.N.R.S. (Directeur et Maître de Recherche)

MM. FRUCHART Robert	Directeur de Recherche
ANSARA Ibrahim	Maître de Recherche
CARRE René	Maître de Recherche
DRIOLE Jean	Maître de Recherche
MATHIEU Jean-Claude	Maître de Recherche
MUNIER Jacques	Maître de Recherche

Je tiens à remercier,

Monsieur le Professeur J.KUNTZMANN, qui a bien voulu me faire l'honneur de présider le jury,

Monsieur le Professeur G.VEILLON, pour la façon dont il a bien voulu m'orienter dans ce travail et m'aider de ses conseils,

Monsieur Ph.JORRAND, Maître de recherche, qui a bien voulu juger mon travail avec beaucoup d'attention.

Je souhaite également remercier Monsieur Michel LUCAS qui a accepté la lourde tâche de me guider tout au long de mes recherches ; ce qu'il a fait avec une compétence et une patience dont je lui suis reconnaissant.

Je n'oublierai pas tous ceux avec qui j'ai travaillé, en particulier Messieurs M.COSNARD, A.LEDUC-LEBALLEUR et F.MARTINEZ pour leur collaboration efficace.

Enfin, je tiens également à remercier le service de tirage pour le soin qu'il a mis à la réalisation matérielle de ce document.

TABLE DES MATIERES

CHAPITRE 3. UN SYSTEME D'INTERPRETATION GRAPHIQUE DE DONNES - ASPECT UTILISATEUR

CHAPITRE 4. LE CHOIX DES MECANISMES GENERAUX ET LEURS IMPLEMENTATIONS

CHAPITRE 5. LES PRINCIPAUX DOMAINES D'APPLICATION
EXEMPLES D'APPLICATION

INTRODUCTION

Les progrès réalisés ces dernières années dans le domaine des techniques graphiques, tant au point de vue de la technologie que du point de vue de la programmation, ont conduit à une extension rapide du nombre de consoles de visualisation en service. De nombreuses publications ([G1],[G2]) concernant les systèmes graphiques ont paru, montrant l'importance de l'activité déployée dans ce domaine.

Un des axes de recherche les plus importants concerne la production d'images dynamiques, c'est-à-dire l'introduction d'une variable supplémentaire qui est le temps. De très nombreuses équipes ou laboratoires travaillent sur le sujet, cherchant à satisfaire une forte demande exprimée aussi bien sur le plan des réalisations pédagogiques ([P1], [P2], [P3], [P4]) ou publicitaires que du pur divertissement (dessins animés proprement dits).

Jusqu'à présent, les différentes recherches effectuées dans ce domaine ont été menées avec comme principal souci la production de films ou de dessins, sans s'attacher à la complexité de description de ces derniers. Il faut donc avoir une bonne connaissance dans les domaines graphique et informatique pour utiliser une console de visualisation, quel qu'en soit le but. Il est évident que cette approche limite sérieusement le nombre d'utilisateurs et n'attire en fait que les personnes ayant pour but de produire des dessins ou des films en tant que tels.

Nous nous trouvons donc constamment en présence de programmes strictement graphiques (les calculs numériques ont pour but de définir un ou plusieurs dessins), pour ne recontrer que très rarement des programmes "mixtes" dont le but principal n'est pas de produire des images. Ceci provient du fait que l'approche informatique choisie ne tient pas suffisamment compte des problèmes des utilisateurs, qui ne sont pas toujours des spécialistes en graphique. Notre expérience prouve que beaucoup d'entre eux ne désirent pas

avoir un système graphique complexe et puissant à leur disposition, mais plutôt un système leur permettant, si besoin est, d'illustrer dynamiquement un programme. Ceci veut dire qu'ils ont écrit et mis au point un programme en utilisant essentiellement des résultats imprimés, et qu'ils souhaitent, pour une raison ou une autre, voir ces résultats sous une autre forme (courbes, histogrammes, dessins quelconques).

Afin d'apporter une solution à ce problème, nous avons été amenés à rechercher les causes de l'existance de ce fossé creusé entre les concepteurs informaticiens et les utilisateurs potentiels. Cette étude nous a permis de dégager les principales classes d'outils graphiques employés, ainsi que les formes sous lesquelles ils apparaissent dans les principaux types de logiciels graphiques (chapitre 1). Dès lors, un premier ensemble d'inconvénients apparaît au niveau de l'utilisation de ces outils, soit de la programmation "classique" des consoles de visualisation. Par la suite, l'étude de la forme générale des systèmes graphiques (chapitre 1) nous a permis de mettre en évidence une deuxième catégorie d'inconvénients, ces inconvénients étant essentiellement affirmés par le type d'utilisation graphique qui nous intéresse.

Cette étude générale, qui nous a permis de déduire les avantages et les inconvénients des principales approches classiques, nous conduit à introduire une classe de systèmes graphiques d'un type totalement nouveau. Cette nouvelle classe étant destinée non pas à surplanter les systèmes existants, mais à combler le fossé existant entre les utilisateurs potentiels et les concepteurs informaticiens. Les différentes remarques que nous avons pu faire, nous ont alors conduits à mettre en évidence la structure de ce type de système ainsi que les divers mécanismes de base à mettre en oeuvre (chapite 2).

Une fois ces divers principes établis, nous avons été naturellement amenés à examiner de plus près les services qu'un utilisateur pouvait attendre d'un tel système, ainsi que la forme sous laquelles ces services étaient exprimables (chapitre 3). Nous avons ainsi défini un ensemble de : règles et de contraintes, le tout étant complété par un langage de description graphique extensible.

La dernière phase, qui consiste à implémenter les mécanismes établis précédemment, nous a conduits à examiner divers domaines tel que le traitement des langages ou l'extensibilité (chapitre 4), et à apporter ainsi certaines modifications à des techniques classiques pour les adapter à nos problèmes.

Nous avons finalement complété l'étude par l'apport de quelques exemples d'applications (chapitre 5), ceci afin de montrer plus clairement les avantages d'un tel système.

Chapitre 1

ASPECT GENERAL DES SYSTEMES GRAPHIQUES

1.1.- GENERALITES SUR LES LOGICIELS GRAPHIQUES CLASSIQUES

1/1.1. Les principales classes d'outils graphiques

Une sortie graphique est généralement constituée d'une composition bien définie de caractères, de points et de segments, avec éventuellement des possibilités d'ombrages par le biais de différentes intensités lumineuses ou encore des possibilités de coloration, ce qui ajoute des paramètres supplémentaires à la description des constituants d'une image. Nous distinguerons trois phases lors de la production d'un dessin:

a/ la définition du dessin dans l'espace utilisateur à l'aide d'un langage de programmation ordinaire et de l'introduction d'un plan image rapporté à un système d'axes de coordonnées.

b/ la création de ce dessin sur l'écran de la console de visualisation à l'aide d'un ensemble d'outils de production de dessin. Ces outils généraux, si l'on se place dans le cas d'un "ensemble graphique" dit de haut niveau, génèrent un véritable programme machine particulier destiné au processeur de dessin. Parmi les diversités des formes sous lesquelles ces outils apparaissent dans les différents logiciels graphiques, nous distinguerons, de par leurs natures différentes, trois catégories principales [G4]:

. les outils de description de dessin qui permettent d'indiquer la composition (caractères, suites de caractères, points, listes de points, segments, listes de segments...) et la position du dessin sur l'écran, ce qui suppose l'introduction d'un système d'axes de coordonnées attaché à l'écran.

. les outils de structuration qui permettent d'établir des liens logiques entre divers composants d'un dessin, soit pour des raisons de descriptions communes (même système d'axes), ou encore pour des raisons de manipulations communes (effacement, transformations...).

. les outils de manipulation du dessin qui permettent de changer la nature (rotation, translation, projection...), soit en modifiant l'image (addition ou suppression d'éléments, coupage, élimination de parties cachées..).

c/ l'introduction du dialogue homme-machine qui permet entre autre de choisir une action parmi un ensemble prévu par le programme d'application. Ces actions étant alors définies à l'aide du langage de programmation employé et d'un ensemble d'outils de dialogue constitué des quatre fonctions de base suivantes:

- la désignation d'un dessin ou d'une partie de dessin,
- l'utilisation d'un "menu",
- la collecte de coordonnées,
- et l'introduction de valeurs.

1/1.2. Les principales classes de logiciels graphiques

Les différents outils graphiques précédents sont plus ou moins bien représentés et d'une manière plus ou moins complexe dans les divers logiciels graphiques. Ceci provient du fait que les uns sont écrits avec un souci constant d'efficacité et de "puissance" (utilisation au mieux des possibilités du matériel) et que les autres sont produits avec comme principal objectif de simplifier la tâche de l'utilisateur et parfois d'assurer la transportabilité (ce qui oblige souvent à ne prendre en compte que les possibilités du matériel le moins complet et ainsi à sous-exploiter les autres dispositifs).

Dans le premier cas, le programme d'application comporte un nombre important d'ordres destinés à gérer l'écran [S1], cette gestion étant faite d'une manière plus ou moins efficace suivant l'utilisateur considéré. Dans le deuxième cas, le nombre d'ordres graphiques diminue d'une manière sensible, parfois aux dépens de l'efficacité, mais toujours en permettant une utilisation plus souple et plus simple [S2].

La complexité d'utilisation des logiciels graphiques de la première catégorie provient du fait que ces derniers reflètent plus l'état de la technologie que celui des besoins des utilisateurs. Ainsi la complexité d'utilisation croît avec les possibilités technologiques de la console graphique considérée.

Les différents outils graphiques énumérés précédemment peuvent être réalisés par le biais d'un langage graphique [S4], d'un langage de commande [S5], d'un système graphique interactif [S6] ou encore d'un ensemble de sous-programmes précompilés utilisables à partir d'un programme écrit à l'aide d'un langage de programmation classique [S7]. La dernière solution est la plus répandue pour diverses raisons telles que la transportabilité et la souplesse d'utilisation [S8]. A ce niveau, deux techniques sont actuellement employées:

- mise à la disposition d'un ensemble de sous-programmes utilisables à partir du programme d'application [S1], [S2].
- intégration de nouveaux éléments syntaxiques dans un langage de haut niveau, nécessitant l'utilisation d'un pré-processeur (ou d'un compilateur modifié) afin de transformer le programme graphique en un programme ordinaire comportant un certain nombre d'appels de sous-programmes [S3]. L'intérêt de la méthode réside dans le fait que l'on peut alors manipuler des variables graphiques au même titre que les autres variables.

1.2.- GENERALITES SUR LES METHODES CLASSIQUES DE PROGRAMMATION DES CONSOLES DE VISUALISATION

Nous nous intéressons donc à la programmation classique des consoles de visualisation à l'aide d'un langage de programmation ordinaire (FORTRAN, ALGOL,..) muni d'un ensemble d'extensions dites graphiques.

Même dans le meilleur des cas, il existe un travail important à effectuer pour incorporer des ordres de visualisation graphique (voir ANNEXE 1) à un programme d'application donné (programme à vocation algorithmique et numérique au départ).

En premier lieu l'utilisateur doit choisir le mode de représentation à employer ainsi que le découpage logique à utiliser afin de définir le (ou les) dessin(s) dans son propre espace (espace utilisateur). Ceci entraîne l'incorporation de nombreux calculs de coordonnées dans le programme d'application.

La phase suivante (création du dessin) est précédée par l'initialisation graphique (initialisation des diverses tables et listes de gestion utilisées par le système, déclaration du découpage logique et des repères de coordonnées employées par le programme...). La production de la liste d'affichage (suite d'ordres élémentaires générés par le système pour éditer le dessin) est ensuite réalisée à l'aide de divers outils graphiques (outils de description de structuration et de manipulation) dont l'emploi est souvent complexe et fastidieux.

De plus un programme graphique ne se conçoit guère sans un minimum d'interactions, ne serait-ce que pour figer une image pendant un certain temps. Ceci implique donc de prendre à nouveau le programme d'application afin d'y incorporer des outils de dialogue.

Il est évident que toutes ces modifications apportées au programme initial peuvent entraîner des erreurs, donc obliger à effectuer plusieurs mises au point. De plus, si la représentation ou le découpage logique adopté ne conviennent pas pour une raison ou pour une autre, c'est encore le programme

lui-même qu'il faudra retoucher (modifications, compilation, mise au point, chargement...). D'autre part, la structure du programme initial est en grande partie masquée par la gestion graphique (voir ANNEXE 1), ce qui est extrêmement génant dans le cas de programmes dont le but n'est pas de produire des images, mais de faire des calculs selon un algorithme donné.

Il serait alors intéressant, tout au moins pour ce genre d'applications, d'avoir non plus des "programmes graphiques" mais des programmes d'applications numériques capables de générer des sorties graphiques à la demande de l'utilisateur. Cette remarque nous amène à imaginer un système graphique, d'un type totalement différent, qui permettrait d'interpréter des données fournies par un programme d'application quelconque, cette interprétation étant faite à l'aide d'outils graphiques étrangers au programme d'application.

1.3.- SYSTEME GRAPHIQUE "GENERAL" ET SYSTEME GRAPHIQUE "D'INTERPRETATION DE DONNEES"

1/3.1. Système graphique "général" et système graphique "spécifique"

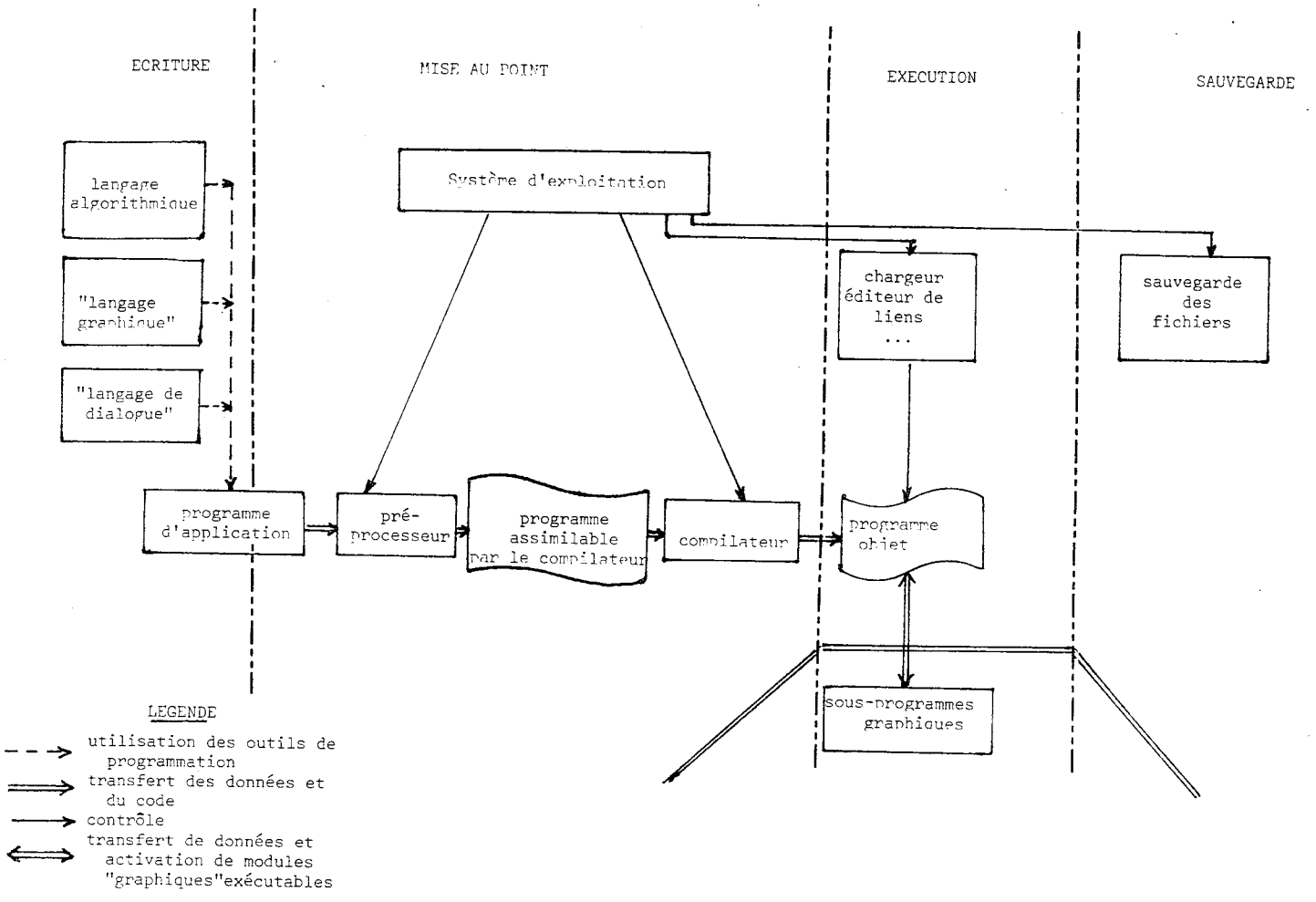


figure 1.1. Système graphique général (aspect utilisateur)

De nombreux auteurs proposent de désigner sous le terme "système graphique" l'ensemble formé par un logiciel graphique et un système d'exploitation, le logiciel graphique étant constitué dans la majorité des cas d'un ensemble de sous-programmes qui gèrent la console de visualisation, tant du point de vue de l'édition que du point de vue du dialogue.

Nous appellerons alors "système graphique général" un ensemble de routines qui gèrent les communications entre l'ordinateur et les dispositifs graphiques, et cela sous le contrôle d'un système d'exploitation quelconque,

capable de "supporter" ce type de terminal (voir figure 1.1.) Notons que dans cette approche, on considère que le "processeur graphique" est suffisamment évolué pour décharger le calculateur de l'affichage proprement dit et de l'entretien de l'image dans le cas des consoles à balayage cavalier (mémoire d'entretien, liste de visualisation ...) [G4], ceci dans le but évident de limiter les échanges entre le calculateur principal et le processeur de dessins, le rôle de chacun étant défini en fonction de la configuration adoptée.

Le nombre important de configurations existantes, tant du point de vue de l'implémentation que du point de vue du matériel, pose le problème de la transportabilité. A l'heure actuelle, un certain nombre de solutions sont proposées aux concepteurs de systèmes graphiques, afin d'assurer une certaine indépendance vis-à-vis du matériel [G3]. Les deux principales approches qui paraissent opposées au départ, mais qui en fait contiennent un certain nombre de similitudes, sont les suivantes:

- l'emploi d'une "console virtuelle" qui est un dispositif fictif par rapport auquel se fait toute programmation au niveau utilisateur [S9],
- et l'implémentation d'un système capable de tenir compte des divers matériels utilisés, en groupant ces dernières par catégories [S10].

Le deuxième type d'indépendance concerne les langages de programmation utilisés. Nous avons vu au paragraphe précédent que cette indépendance était en grande partie réalisée par l'emploi de sous-programmes précompilés. Cet emploi est évidemment subordonné à la compatibilité existante entre des sous-programmes écrits à l'aide de langages de programmation différents, compatibilité qui dépend essentiellement du système d'exploitation utilisé, mais aussi de la classe des langages considérés (type de syntaxe, niveau de langage, structure de données...).

La dernière forme d'indépendance, qui nous intéresse plus particulièrement, concerne le type d'application. Cette indépendance est en partie réalisée dans les systèmes graphiques généraux par le fait que la programmation est faite à l'aide d'un langage de programmation ordinaire et d'un ensemble d'outils graphiques, le principal obstacle résidant alors dans le mode

d'utilisation de ces outils. Par opposition, nous trouvons le système graphique spécifique utilisable à partir d'un langage de commande strict, excluant ainsi tout emploi d'un autre langage de programmation et interdisant de ce fait toute évolution du logiciel impliqué. Ceci signifie qu'un système de ce type est dévolu à une application particulière (système UNISURF pour la conception de carrosseries automobiles chez RENAULT par exemple).

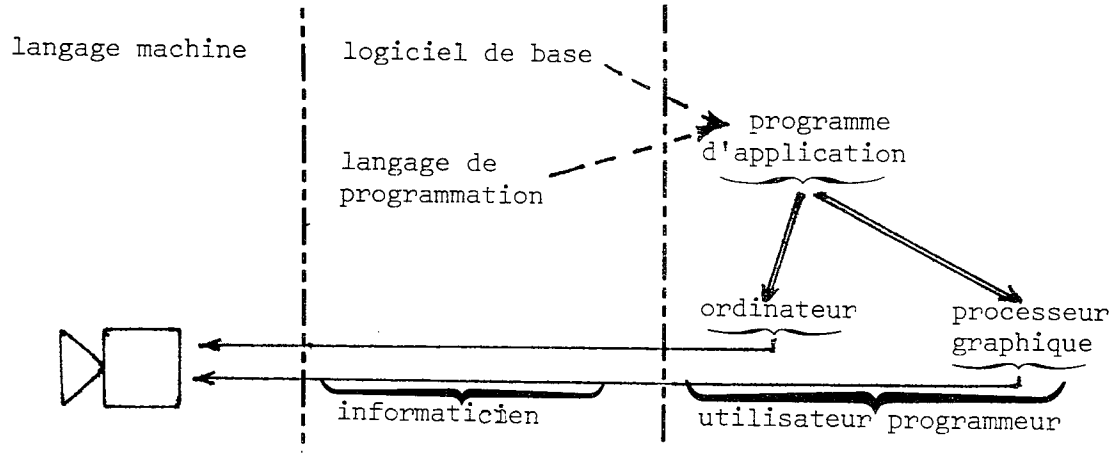


figure 1.2.: système graphique "général"

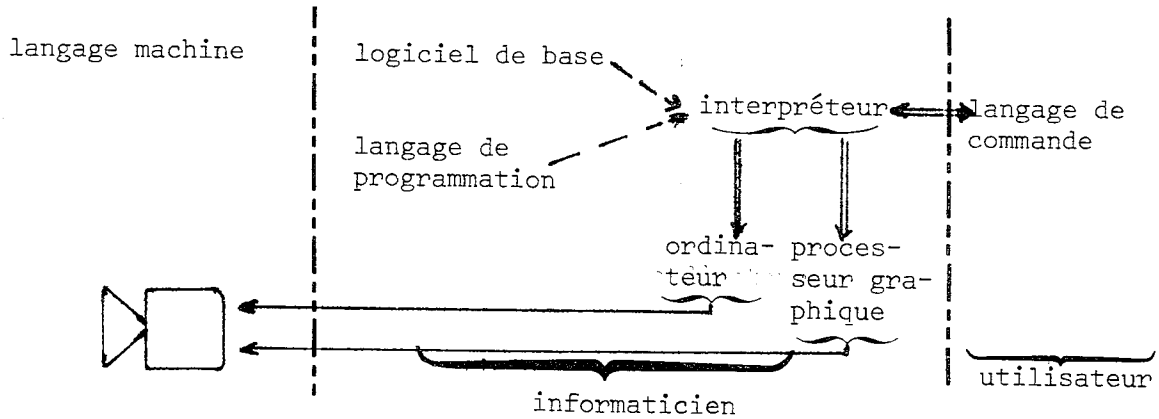


figure 1.3.: système graphique "spécifique"

LEGENDE:

- ----> utilisation des outils de programmation
- ⟹ transfert de code source
- ⟶ transfert de code objet
- ⟺ transfert d'ordres graphiques

Afin d'accroître les possibilités d'indépendance vis-à-vis de l'application tout en conservant les avantages de simplicité d'emploi relatifs aux systèmes graphiques spécifiques, nous introduirons un nouveau type de système: "le système graphique d'interprétation de données".

1/3.2. Système graphique "d'interprétation de données"

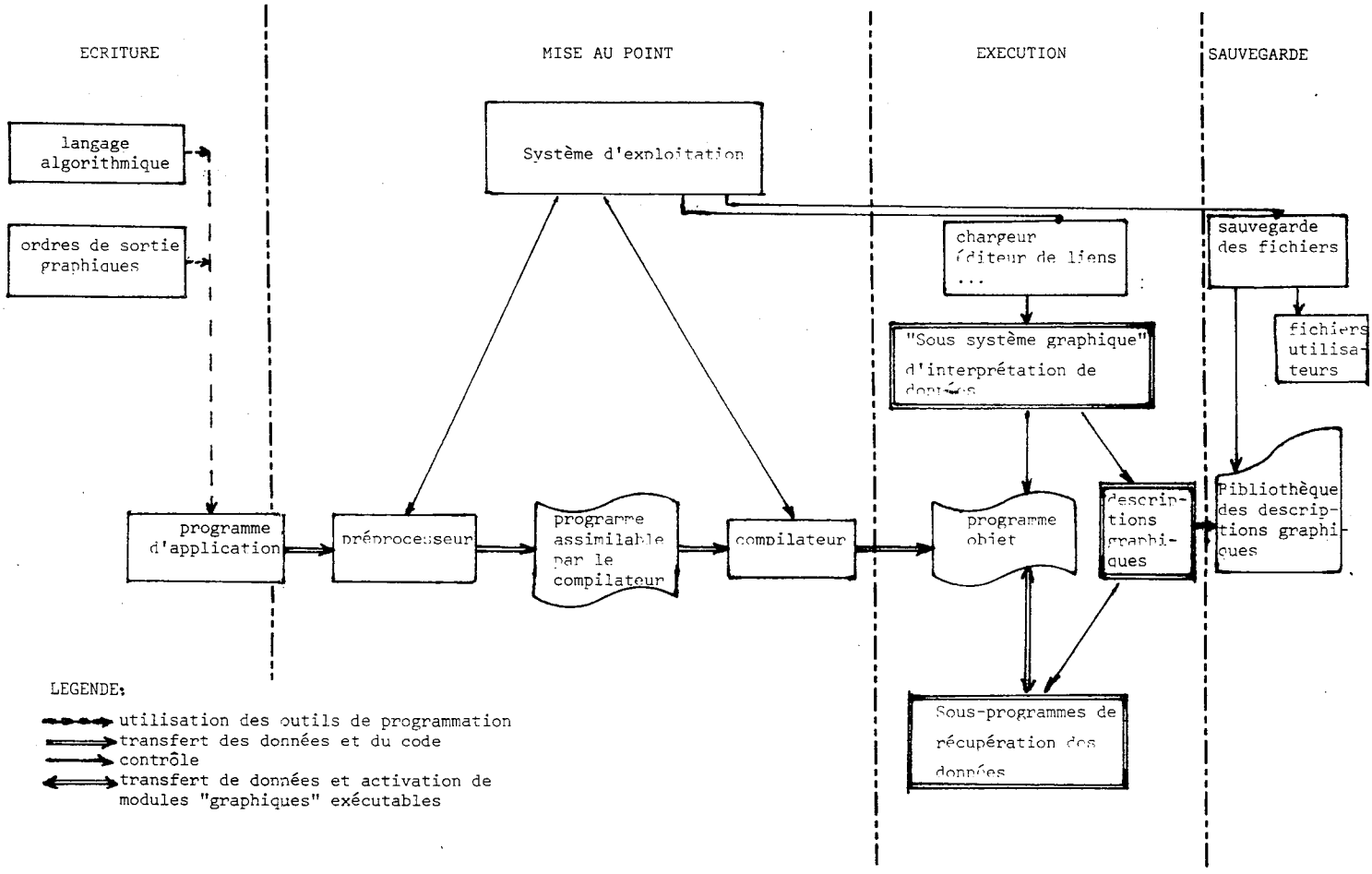
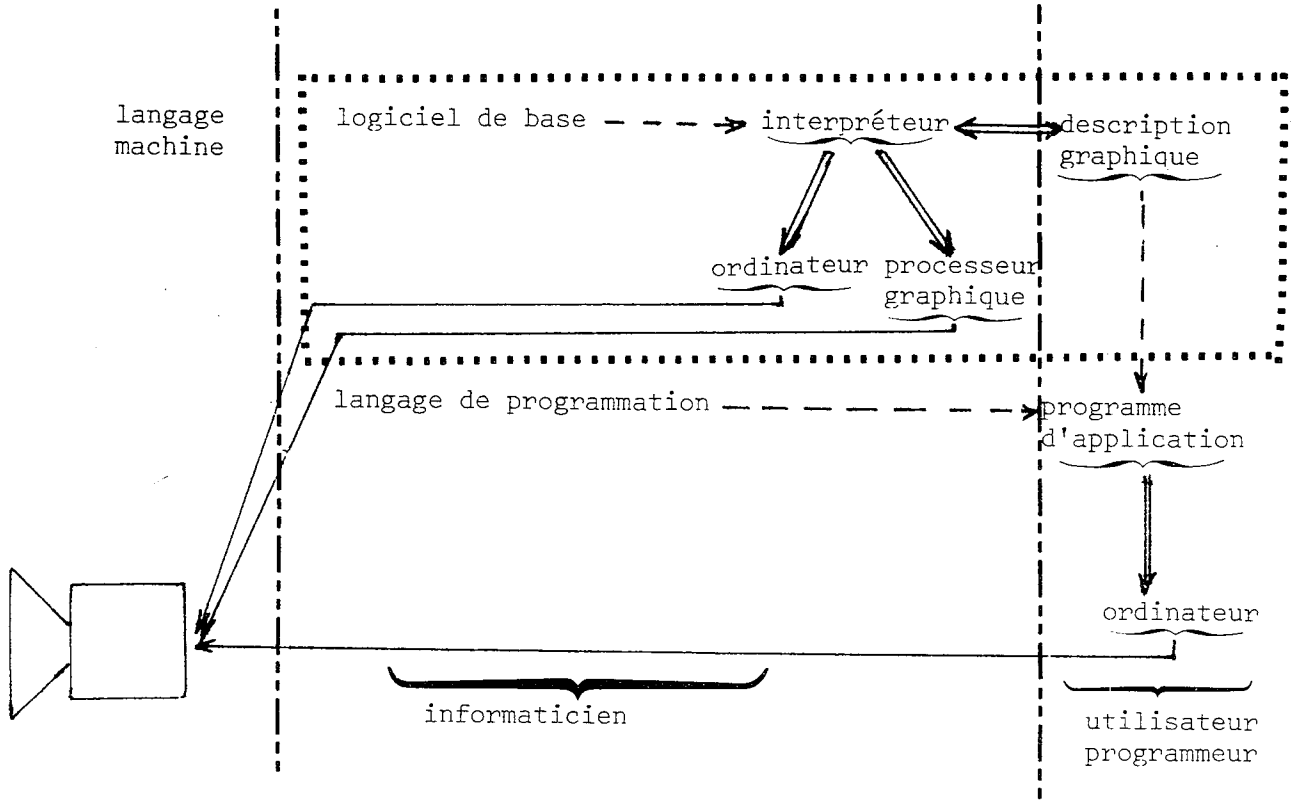


figure 1.4: Système graphique d'interprétation de données (aspect utilisateur)

Cette catégorie de système s'efforce de réaliser l'indépendance vis-à-vis du type d'application, et cela aux dépens de la généralité (voir figure 1.4) Cette indépendance étant alors réalisée par la séparation du programme d'application et de la description graphique.



LEGENDE:

- — - → utilisation des outils de programmation
- ⟹ transfert de code source
- ⟶ transfert de code objet
- ⟺ transfert d'ordres graphiques

figure 1.5.: Système graphique 'd'interprétation de données"

Nous remarquerons dans ce schéma que l'ordinateur est utilisé à deux niveaux différents (interprétation et programme d'application). Or l'interprétation, qui est liée à la gestion graphique, se rapproche de par sa nature, du processeur graphique. Il parait alors intéressant d'accentuer le découpage initial, en introduisant un calculateur satellite qui prendrait en charge la gestion graphique (interprétation, processeur de dessin...). Le problème étant alors de définir le rôle de chacun, afin de minimiser les échanges entre calculateur satellite et calculateur principal.

A priori, rien n'empêche d'utiliser ce type de système d'une manière générale. Cependant, au cours de l'étude qui suit, nous mettrons à jour un certain nombre de lacunes qui, à l'heure actuelle, limitent l'utilisation d'un tel

système à la catégorie d'application appelée: "Illustration dynamique de programmes". Le principal problème étant posé par l'intégration du dialogue homme-machine au niveau d'un programme d'application dit indépendant (cf.§2.3.2 Notons enfin, que cette indépendance relative est payée au prix d'une perte d'efficacité, du fait qu'il y a intégration d'un interpréteur graphique.

Chapitre 2

UN SYSTEME D'INTERPRETATION GRAPHIQUE DE DONNEES

- ASPECT SYSTEME -

2.1.- UN SYSTEME D'INTERPRETATION GRAPHIQUE DE DONNEES

Nous nous trouvons ici en présence d'utilisateurs qui ont écrit et mis au point un programme en utilisant essentiellement des résultats imprimés et qui souhaitent "voir" ces résultats sous une autre forme. La remarque fondamentale est qu'en fait, ils ne veulent rien changer au programme lui-même, et qu'il souhaitent simplement interpréter autrement des tableaux de résultats qu'ils ont fait calculer auparavant. A l'heure actuelle, tout ce qu'on leur propose est d'utiliser les techniques exposées au paragraphe précédent, ce qui conduit à décourager beaucoup d'utilisateurs potentiels.

Cette situation provient du fait que l'on mélange deux choses:

- le programme d'application, chargé de calculer des valeurs numériques et de les ranger (par exemple) dans des tableaux,
- le programme d'interprétation de ces valeurs numériques, chargé de les mettre en forme pour une exploitation.

Nous nous proposons d'étudier un système d'illustration dynamique de programmes permettant dans la mesure du possible, d'éviter les manipulations au niveau du programme d'application. Ceci signifie que, partant d'un programme ordinaire ayant la configuration habituelle suivante (existant dans à peu près tout langage de programmation):

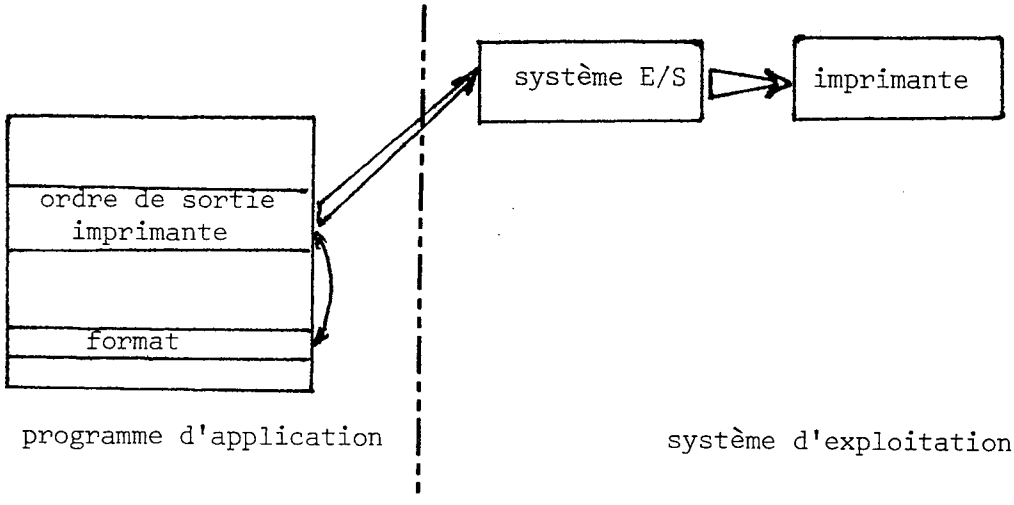


figure 2.1.: configuration courante

nous voulons interpréter graphiquement les sorties déjà prévues par un certain nombre d'ordres d'écriture (que nous noterons "write"). La seule manipulation que nous voulons autoriser consiste à transformer les ordres de "sortie imprimante" en ordres de "sortie graphique" éventuellement assortis d'un ou deux paramètres. Nous ne voulons en aucun cas modifier la structure du programme, tout se passant comme si l'on faisait une opération d'entrée/sortie ordinaire, cette opération étant toutefois soumise à un certain contrôle de la part de l'utilisateur. Nous obtenons alors le schéma suivant:

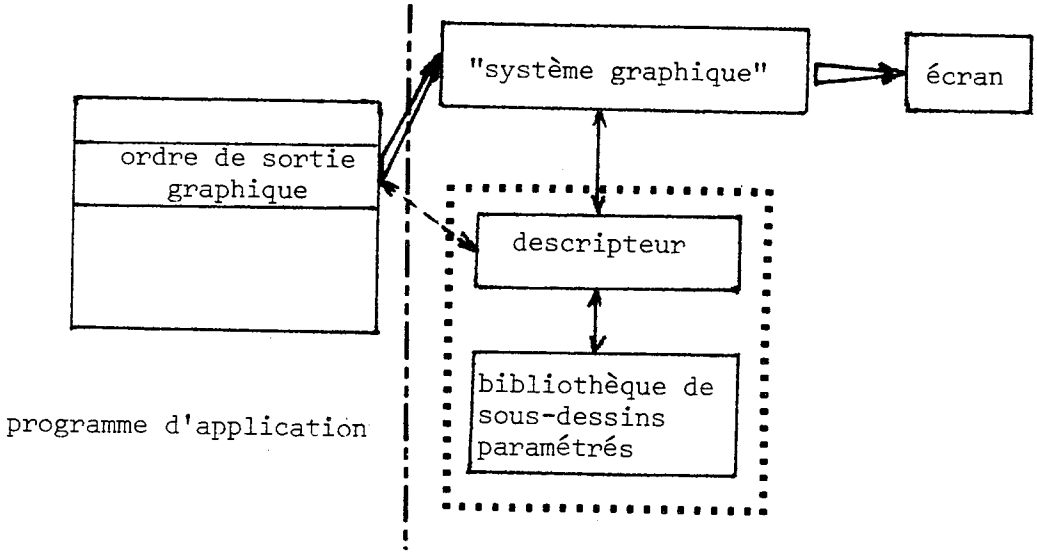


figure 2.2.: nouvelle configuration

L'édition graphique se fera alors au niveau d'un "système" et d'un ensemble de descripteurs d'images non inclus dans le programme d'application, séparant ainsi la gestion de l'image de l'algorithme de calcul.

Au niveau de la description de l'image nous utiliserons alors un ensemble graphique possédant les primitives suivantes:

- les primitives de description du dessin, qui font référence à de véritables sous-dessins paramétrés (les données fournies par le programme d'application interviennent dans la paramétrisation de ces sous-dessins).
- les primitives de structuration du dessin.

- les primitives de manipulation du dessin, qui permettent de modifier le dessin originel sans en changer la nature (affinité, translation). Il n'existe pas toutefois de primitives de manipulation telles que l'addition ou la suppression d'éléments, le coupage... La gestion de l'image proprement dite (déclaration de figures, définitions de repères et de types de coordonnées, options de coupages, ordres d'affichage et d'effacement, mise à jour...) est en effet prise en charge par le système en fonction de divers paramètres tels que l'état des données ou l'encombrement momentané de l'écran.

Nous remarquerons l'absence de primitives de dialogue à ce niveau. Ceci provient du fait que nous sommes en présence d'un langage de description graphique, et que le dialogue se place en fait au niveau du programme d'application (voir § suivant).

2.2.- LES MECANISMES DE BASE

a/
Il existe au niveau de système un mécanisme permettant de récupérer:

- les numéros de descripteurs,
- les adresses des différents paramètres à visualiser.

Notons que ce mécanisme existe dans tous les langages de programmation (conventions d'appel de sous-programmes ou de procédures) et qu'il permet déjà d'utiliser, à partir d'un langage quelconque, des sous-programmes précompilés écrits dans d'autres langages (utilisation classique de sous-programmes assembleur en PL/1 ou de sous-programmes FORTRAN en ALGOL W). Ceci veut dire que l'on peut à priori accepter en entrée n'importe quel programme compilé, quel que soit le langage d'écriture. La seule manipulation consiste à compiler correctement les ordres de sorties graphiques, ce qui peut se faire au prix d'un pré-traitement relativement simple.

Cette précompilation qui est spécifique au langage de programmation utilisé, permet de transformer les ordres de sorties graphiques en appels de sous-programmes externes compatibles avec le langage.

b/
Le système dispose d'une bibliothèque de descripteurs standards évitant à l'utilisateur de définir lui-même la sortie graphique qu'il désire. On trouve par exemple: interprétation sous forme de tableau numérique, sous forme de courbe, sous forme d'histogramme, etc... Ceci permet par exemple de vérifier l'allure des résultats d'un calcul sans avoir à se préoccuper de toute la partie graphique: il suffit de remplacer les ordres "write" par des ordres "écran" assortis d'un numéro de descripteur.

c/
L'utilisateur peut enrichir la bibliothèque d'éléments, afin de créer des interprétations différentes de celles proposées par le système. Ceci veut dire que l'on dispose, au niveau du système graphique, de moyens pour spécifier comment doit se former le dessin à partir des valeurs numériques envoyées par le programme d'application (définition de dessins-type paramétrés).

d/
L'utilisateur peut changer à tout moment un descripteur, c'est-à-dire modifier l'interprétation d'une série de valeurs. Ceci se fait à nouveau grâce au système graphique, sans retoucher au programme d'origine.

Le système procure donc à l'utilisateur les avantages suivants:

- facilité de mise en oeuvre de l'illustration d'un programme (à la limite, simple appel du système graphique si l'on se contente des représentations standards),
- possibilité d'étudier plusieurs représentations pour un même programme, sans toucher au programme, en modifiant simplement les descripteurs,
- possibilité d'illustrer une série de programmes à l'aide d'une même représentation (programme de tri par exemple) en se contentant de changer les programmes en entrée du système, une fois la représentation de base choisie.

Il faut noter à ce niveau que si l'on peut rejeter la partie "interprétation graphique" hors du programme d'application dans de nombreux cas (en particulier, pour tous les programmes de simulation), le problème n'est pas aussi simple en ce qui concerne le dialogue homme-machine. Il est clair que l'on peut avoir une approche analogue si l'on se contente de dialogues n'intervenant pas sur le cours du programme, (pas de menu) mais se contentant de modifier certaines valeurs de paramètres. On pourra alors de nouveau demander l'obtention de ces valeurs à partir du programme d'application, sans avoir à spécifier comment elles sont obtenues. C'est au niveau du système graphique que l'utilisateur décidera, en fonction des disponibilités, de l'outil qu'il emploiera. Cependant, cette philosophie est mise en défaut à partir du moment où l'on veut intervenir sur le déroulement du programme proprement dit: il faut avoir accès à des identificateurs permettant de réaliser des ruptures de séquence, ce qui n'est possible dans la plupart des langages qu'en prévoyant dans le programme proprement dit ces ruptures afin que la compilation permette de générer un code correct (cas de FORTRAN, ALGOL, PL/1...). Cependant, on peut remarquer que bien souvent, en particulier en ce qui concerne les simulations, le dialogue est assez pauvre et correspond à une

simple modification des données sans modifier le cours du programme. Nous nous contenterons donc dans un premier temps, de ne permettre que des actions cataloguées intervenant au niveau du dessin même (taille, forme, position,...) ou au niveau de l'enchaînement dynamique des images.

2.3.- STRUCTURE DU "SYSTEME" ET LIENS AVEC LE PROGRAMME D'APPLICATION

2/3.1. Structure de la partie "génération de dessin"

Nous allons dans ce paragraphe dégager les divers modules nécessaires à la réalisation du système de ce type, et établir ainsi le rôle exact de chacun ainsi que la forme des liaisons à introduire.

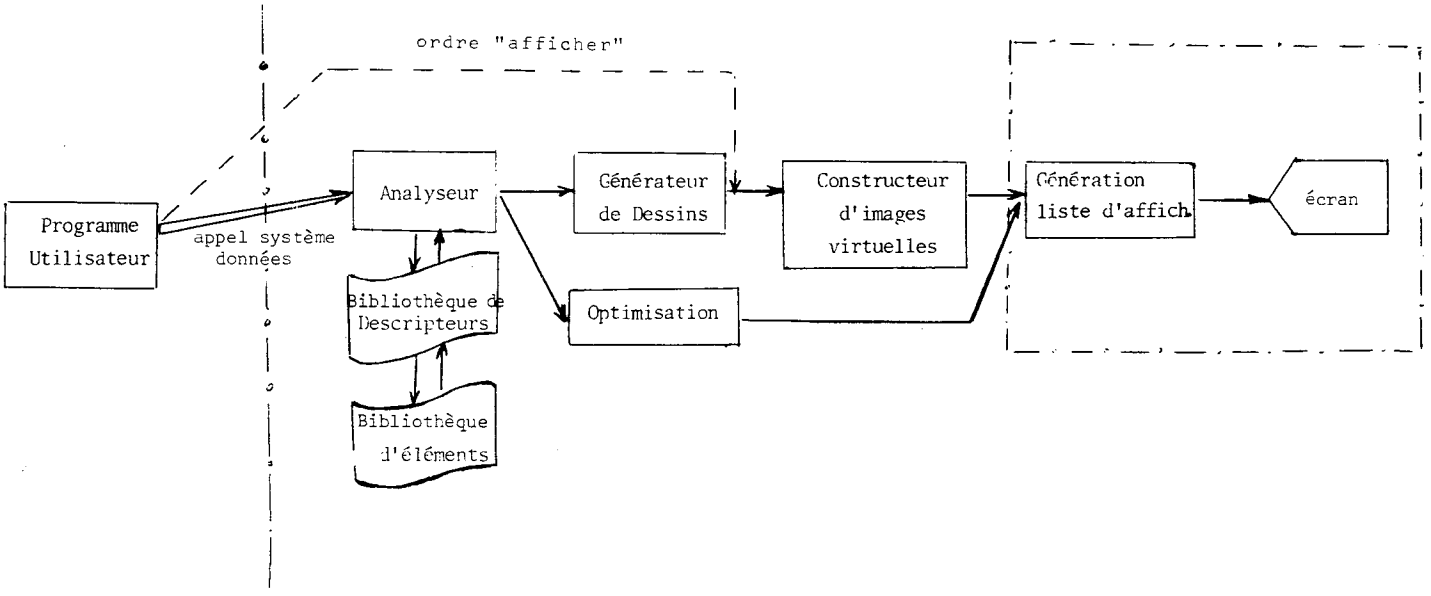


figure 2.3.: structure générale du "système"

Analyseur

Il décode les descripteurs contenus dans la "bibliothèque de descripteurs" à la demande du programme utilisateur et produit un code interne qui sera interprété par le "générateur de dessins". Il travaille d'une manière analogue à l'analyseur lexicographique et à l'analyseur syntaxique d'un compilateur, le langage à compiler étant dans notre cas le langage de description graphique.

Générateur de dessins

Il engendre "virtuellement" les dessins correspondant au code interne produit par l'anlyseur, tout en faisant le lien avec les données fournies par le programme d'application. Le code produit représente les dessins sous forme de listes de nombres "non dimensionnés" que nous nommons trames des dessins. Il reste alors à donner une taille et une position sur l'écran à chacun de ces dessins.

Constructeur d'images virtuelles

Il construit une image sur l'écran virtuel à l'aide des "trames" des dessins obtenus précédemment. Pour cela, il donne une taille à chaque dessin en faisant subir une affinité positive d'axe Oy (axe des ordonnées) de direction Ox (axe des abscisses) et de rapport k1 et une affinité positive d'axe Ox, de direction Oy et de rapport k2, à la "trame" correspondante.

Le constructeur d'images virtuelles affecte ensuite une portion d'écran virtuel à chaque dessin, ce qui se traduit par le calcul des rapports k1 et k2 des affinités précédentes et par le calcul des constantes k3 et k4 de translation horizontale et verticale.

Optimisation

Le système décide si il est possible de "mettre à jour" l'image précédente afin d'éviter d'en construire une nouvelle. Cette mise à jour ne s'effectue pas au niveau du terminal graphique car elle dépendrait alors du matériel utilisé. (Elle serait, par exemple, possible sur un terminal graphique IBM 2250 qui possède une mémoire d'entretien que l'on peut manipuler, mais impossible sur un terminal TEKTRONIX qui fonctionne avec un tube mémoire).

Cette"mise à jour" se situe donc uniquement au niveau de la construction de l'image par le système. Il est en effet inutile de rebatir une image contenant à quelques différences près, les mêmes éléments que la précédente.

Génération de la liste d'affichage

Ce module dépend directement de la console graphique utilisée. Il génère le code nécessaire à la production de l'image sur le terminal graphique utilisé.

Bibliothèque de descripteurs

Elle contient la "description" des images qui composent les films. Le langage utilisé est essentiellement un langage de manipulation d'éléments graphiques dont les "modes" sont définis par un ensemble extensible de mots clés. Ces mots clés font implicitement référence à des descriptions graphiques d'éléments de base, ces descriptions de niveau inférieur étant stockées dans la "bibliothèque d'éléments". Ce mécanisme se rapproche de celui utilisé par le langage SIMULA 67 ([L1], [L23]) pour décrire le pseudo-parallélisme à l'aide d'objets que l'on crée à partir de "classes" définies auparavant (cf § 3.2.3.).

Bibliothèque d'éléments

Elle contient la description graphique des éléments de base que le système est capable de générer. Ces descriptions, qui correspondent aux déclarations de classes de SIMULA 67, définissent des "types de dessins" référenciés par les mots clés précédents (mots clés du langage de description d'images de la bibliothèque de descripteurs). Toutes les descriptions de type d'élément qui sont stockées dans cette bibliothèque sont en fait des modules exécutables capables de générer des "trames" de dessins, en fonction de cartains paramètres (cf § 3.2.3.).

2/3.2. Structure de la partie interactive

Nous ne traiterons dans ce paragraphe que l'aspect "structure" de l'interaction, sans développer la forme des fonctions mises en oeuvre. En effet, l'intérêt d'introduire telle fonction ou telle autre n'est apparent que lors de l'étude du langage de description graphique.

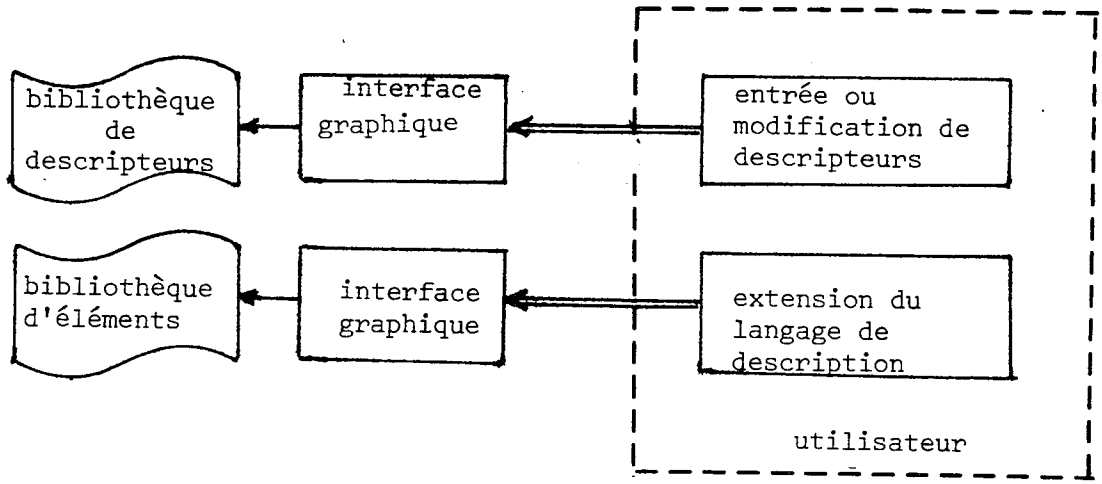
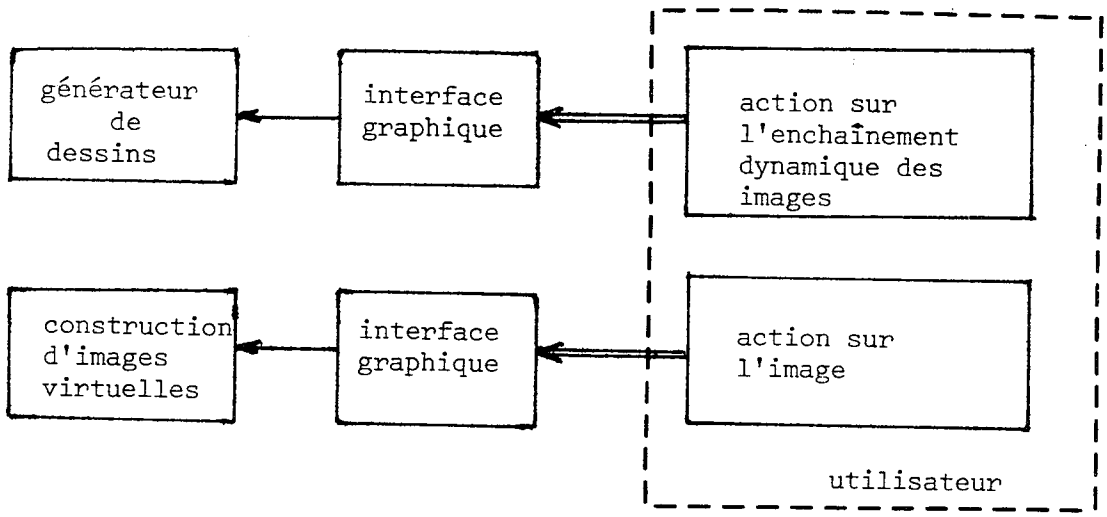


figure 2.4.: action sur le mode de représentation



LEGENDE :

⟹ transfert de données et d'ordres graphiques
⟶ transfert de code interne

figure 2.5. Action au niveau des images

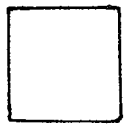
2/3.3. Liaisons avec le programme d'application

2/3.1.1. Définitions

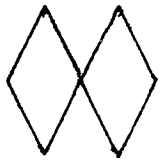
(1) On appelle mode de base un sous-dessin paramétré qui possède une description dans la bibliothèque d'éléments, ce sous-dessin pouvant être constitué à partir de plusieurs autres sous-dessins (cf § 3.2.3.1.).

exemples:

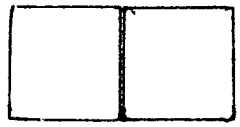
mode M1



mode M2



mode M3
construit à partir du mode M1



mode M3
construit à partir du mode M2

(2) On appelle unité graphique la description d'un dessin ou d'un ensemble de dessins ne faisant intervenir qu'un seul mode de base défini dans la bibliothèque d'éléments. Cette unité graphique ne peut être décomposée du point de vue de la représentation par le système, et chaque unité graphique correspond à la représentation d'une unité de donnée (une ou plusieurs données groupées).

(3) On appelle bloc de données un ensemble de données groupées en vue d'une représentation globale.

(4) On appelle valeur, la valeur entière, réelle, booléenne..., représentée par une donnée de type simple.

2/3.3.2. Structuration de données

Certaines données n'ont de singification, tout au moins en ce qui concerne l'interprétation, que lorsqu'elles sont mises en relation avec d'autres (un couple de coordonnées pour un point d'un plan par exemple). Il est donc nécessaire de pouvoir grouper ces données en "blocs de données" afin de les associer aux unités graphiques correspondantes.

Deux possibilités sont alors offertes:

- le nombre de "valeurs", dont a besoin l'unité graphique, est connu implicitement ou explicitement par le système par l'intermédiaire des descripteurs. Le blocage de données est alors automatique.

- le nombre de "valeurs" associables à l'unité graphique est variable. Dans ce cas, le programme d'application doit spécifier le groupement des données (cf.§ 3.2.2.).

Le regroupement des données est donc fait, soit automatiquement par l'intermédiaire de la structuration des données propre au programme d'application (tableaux par exemple), soit sur spécification au niveau de la conception des sorties graphiques. Ce dernier type de structuration s'adressant directement à l'interprétation graphique des données.

2/3.3.3. Choix des valeurs

Il s'agit de fournir un mécanisme permettant de choisir un sous-ensemble de valeurs dans un ensemble fourni par le programme d'application, afin de ne représenter momentanément que les valeurs ayant un sens à l'instant considéré. Ceci peut être utile, par exemple, pour ne représenter dans un tableau possédant un grand nombre d'élements, que le sous-ensemble ayant un intérêt à un instant donné.

2/3.3.4. Correspondance entre les données et la description graphique

Le programme d'application fournit au système des données, groupées ou non, accompagnées d'un numéro de descripteur. Chaque bloc de données est alors affecté au descripteur correspondant suivant le schéma ci-après:

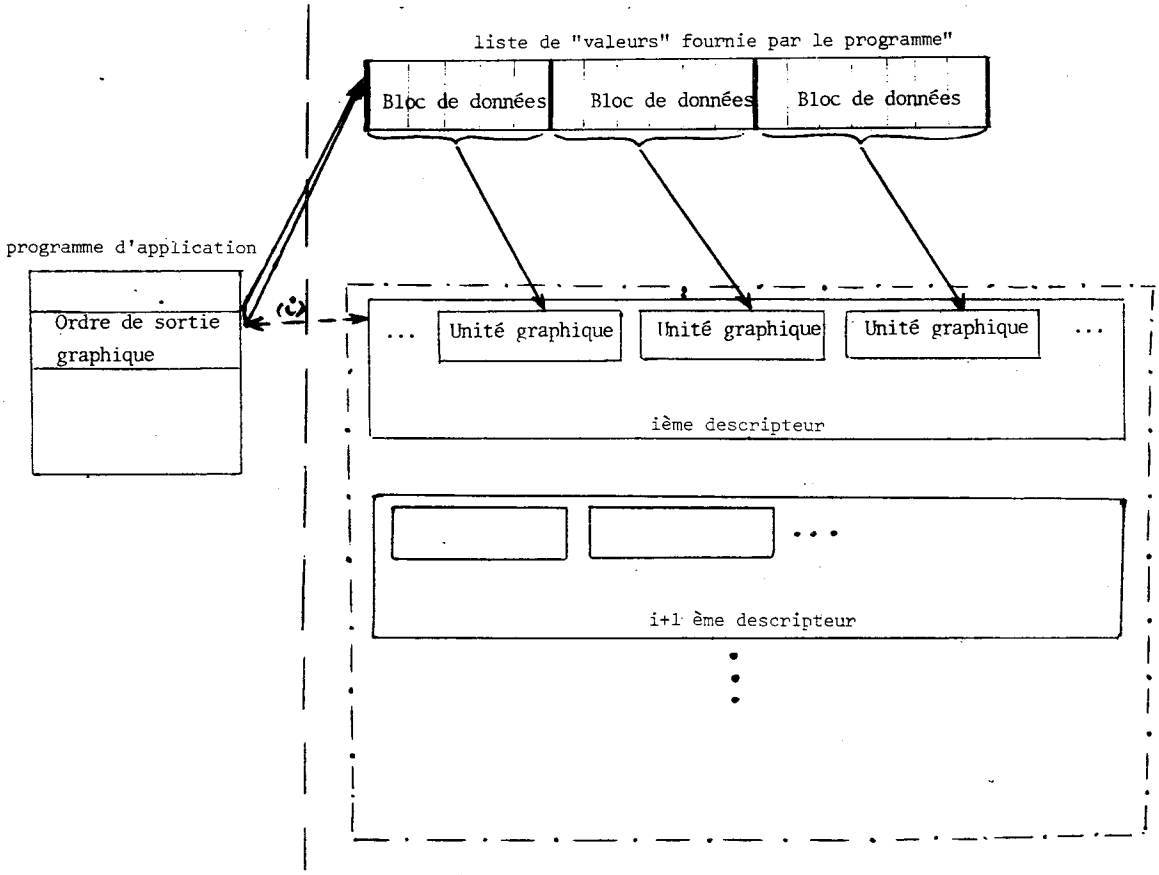


figure 2.6.: correspondance entre les données et les descriptions graphiques

Notons que chaque unité graphique est traitée de manière séquentielle dans un descripteur donné et que tout se passe comme si un descripteur bouclait indéfiniment sur lui-même (retour au début lorsqu'il se termine).

Remarque : La méthode par laquelle les données sont obtenues n'intervenant pas, le système peut aussi bien "puiser" les données à partir d'un programme d'application qu'à partir d'un fichier quelconque. Notons cependant, que cette indépendance vis-à-vis de l'application et du langage de programmation employé, est subordonnée à la cohérence du système d'exploitation en ce qui concerne les conventions de liaisons entre les sous-programmes.

2/3.3.5. Construction des images

Nous appelons image le groupement, à un instant donné, d'un ensemble de représentations graphiques.

Nous avons au départ le choix entre deux possibilités:

- gérer totalement les images, indépendamment du programme d'application (par les descripteurs),
- laisser au programme le droit de constituer lui-même ses images.

La deuxième solution semble être la meilleure dans la majorité des cas, car le programme seul est capable de dire à un instant donné si la totalité des données relatives à une image a été fournie au système.

2.4.- GESTION GRAPHIQUE

Toujours dans le but de faciliter l'utilisation des terminaux graphiques, nous proposons de fournir une option système qui permette de décharger l'utilisateur de toute la partie fastidieuse consistant à placer les éléments graphiques sur l'écran (calcul de taille et de position). Bien entendu certaines décisions prises par le système peuvent aller à l'encontre des désirs de l'utilisateur. Il est alors nécessaire de pouvoir intervenir soit par le biais des descripteurs, soit par simple "action graphique cataloguée" (à l'aide d'outils de dialogue), afin d'imposer sa propre configuration.

2/4.1. Taille des divers constituants de l'image

La taille des éléments est calculée par le système au moment de l'édition. Chaque élément (sous-dessin paramétré) est décrit par le système dans son propre repère d'axes et possède une taille minimum qui est fonction, d'une part du dessin de base, et d'autre part du mode de variation des divers composants sout l'action de la (des) valeur(s). Cette taille minimum correspond à l'espace écran minimum nécessaire à la création du dessin correspondant (surface minimum nécessaire pour afficher une chaîne de caractères de longueur donnée par exemple). La taille réelle des éléments peut être alors, soit fournie par les descripteurs, soit calculée par le système en fonction de l'encombrement de l'écran à l'instant considéré.

2/4.2. Allocation d'espace-écran

L'allocation espace-écran aux divers constituant de l'image peut être faite de deux manières différentes:

- soit à l'aide d'indications fournies au système par l'intermédiaire des descripteurs,
- soit en laissant la gestion de l'écran au système, en se réservant toutefois la possibilité d'agir directement sur l'image par l'intermédiaire des primitives d'interaction.

L'espace-écran est alloué par blocs rectangulaires aux différentes figures qui composent l'image. Cette allocation est effectuée dans un premier temps à partir d'un écran fictif dont les dimensions dépendent de la taille et de

la surface couverte par les figures décrites en "taille minimum". Ce procédé est employé afin de pouvoir découper la surface de l'écran en zones dont les tailles sont peu différentes de celles des figures correspondantes (surfaces d'écran minimales nécessaires à la génération de ces figures). Ainsi, quel que soit l'algorithme de placement choisi, on obtient une occupation satisfaisante (suppression des surcharges localisées et des espaces vides) d'un écran fictif dont les dimensions évoluent avec la taille des éléments à placer. Le fait de ramener l'ensemble aux dimensions réelles permet ensuite de calculer la taille réelle des divers constituants de l'image.

Cependant le problème déjà non trivial posé par la répartition de blocs géométriques sur une surface plane rectangulaire ne reflète qu'une partie du problème réel. En effet, il entre en jeu d'autres critères dits "visuels" qui sont difficilement exprimables sur le plan mathématique car ces derniers sont souvent subjectifs.

Nous arrivons donc rapidement à la conclusion qu'il n'existe pas une solution satisfaisante mais des solutions suceptibles d'être satisfaisantes. C'est dans cette optique que le système propose des configurations possibles, selon un algorithme qui n'est certainement pas optimal mais qui offre une solution au problème. En dernier lieu, c'est donc le seul juge possible, à savoir l'utilisateur, qui tranchera à l'aide des primitives de dialogue.

Si nous nous plaçons dans l'optique d'un système puissant, capable d'offrir une gestion automatique optimale de l'écran, il est évident que les solutions proposées sont insuffisantes. Si par contre nous nous trouvons en présence d'un système plus modeste dont le but est de fournir des représentations graphiques à des données, la gestion automatique de l'écran prend en fait un aspect d'aide à l'utilisateur. Nous nous plaçons exactement dans la situation d'un utilisateur qui désire effectuer des sorties imprimantes à l'aide d'un langage qui possède des ordres "write" avec ou sans format de sortie (ALGOL W par exemple). Si l'utilisateur se contente d'une mise en page automatique simple (pas d'espace, pas de saut de ligne, écriture en début de ligne) il utilise un ordre de sortie non formatée. Si par contre

il désire une mise en page plus ou moins complexe, il décrit ses sorties à l'aide d'ordres de sorties formatées. Nous remarquerons toutefois que dans le cas du système d'interprétation graphique de données, l'aspect rigide de la première solution (sorties non formatées) disparait en partie grâce à l'apport des moyens d'interaction.

Notons enfin que si l'on se place dans ce contexte, il devient intéressant de pouvoir enregistrer les diverses options choisies par l'utilisateur afin de pouvoir restituer par la suite les configurations adoptées.

Chapitre 3

UN SYSTEME D'INTERPRETATION GRAPHIQUE DE DONNEES

ASPECT UTILISATEUR

3.1. FORMALISME GRAPHIQUE

3/1.1. La structuration graphique

3/1.1.1. Les éléments

Ce sont les unités de base de programmation de la console de visualisation employée. Les éléments sont dont inaccessibles au niveau du langage graphique utilisé.

3/1.1.2. Les objets

Ce sont les unités de base, accessibles à l'aide du logiciel graphique employé. Dans le cas du système d'interprétation graphique de données, ces primitives correspondent à de véritables sous-dessins.

a) Définitions préliminaires :

(1) On dit qu'un dessin subit une déformation lorsque la forme, la taille ou la position de ce dessin varie.

(2) On dit qu'un dessin subit une transformation lorsque ce dessin est l'objet d'une modification quelconque, cette modification étant obtenue par déformation du dessin ou par adjonction (ou suppression) d'un symbole ou d'un sous-dessin.

(3) On appelle algorithme de transformation un algorithme permettant d'appliquer une transformation à un dessin, et ceci en fonction d'un ensemble de paramètres.

(4) On appelle vecteur de valeurs un ensemble ordonné de valeurs, ces valeurs caractérisant les données à visualiser (cf § 2/3.3.2.).

(5) On appelle dessin de base un dessin prédéfini utilisable par le système. Ce dessin n'est pas directement représentable et doit subir une transformation pour donner naissance à un dessin représentable.

. notations :

B = {ensemble des dessins de bases}

A = {ensemble des algorithmes de transformations}

V = {ensemble des vecteurs de valeurs}

T = {ensemble des transformations}

Soit T une application de l'ensemble produit $A \times V$ dans l'ensemble T.

$$T : A \times V \rightarrow$$
$$(A, V) \rightarrow T(A, V) = t$$

(6) On appelle transformation nulle T_o, la transformation qui correspond à un couple (A, Ø) de $A \times V$, l'algorithme A étant quelconque. Cette transformation ne modifie pas le dessin sur lequel elle s'applique.

$$T : A \times V \rightarrow T$$
$$(A, \emptyset) \rightarrow T_o = T(A, \emptyset)$$

(7) On appelle objet potentiel un couple (B, A) de l'ensemble produit $B \times A$.
Soit un couple quelconque (A, V) appartenant à l'ensemble produit $A \times V$, et un dessin de base B quelconque appartenant à B.

(8) On appelle objet un dessin obtenu en appliquant une transformation T(A, V) à un dessin de base B. On note {B, A, V} cet objet.

$$B \in B \xrightarrow{T(A, V)} \{B, A, V\} \in \text{ ensemble des objets.}$$

(9) On appelle objet vide un dessin obtenu en appliquant la transformation nulle T_o à un dessin de base B.

$$B \in B \xrightarrow{T(A, \emptyset)} \{B, A, \emptyset\}$$

L'opération qui permet de passer de l'objet vide à l'objet tel qu'il a été défini précédemment est appelée valorisation d'objet. Cette opération consiste à appliquer la transformation T(A, V) au dessin de base B.

$$\{B, A, \emptyset\} \xrightarrow{\text{"valorisation"}} \{B, A, V\}$$

b) Objets à bases statiques :

Soit A' l'ensemble des algorithmes de transformations qui ne font subir aucune déformation aux dessins sur lesquels ils s'appliquent. Le rôle de ces algorithmes étant de rajouter une information au dessin par l'adjonction d'un symbole ou d'un sous-dessin.

$$A' \subset A$$

Soit un couple quelconque (A', V) appartenant à l'ensemble produit $A' \times V$, et un dessin de base B quelconque appartenant à B.

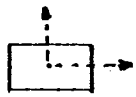
(10) On appelle objet à base statique un dessin obtenu en appliquant une transformation T(A', V) à un dessin de base B (le dessin de base est indéformable pendant toute la durée de vie de l'objet).

$A' \in A'$, $B \in B \xrightarrow{T(A, V)}$ {B, A', V} $\in$ ensemble des objets à bases statiques

Remarque : Un objet vide {B, A', Ø} est donc représenté dans ce cas par le dessin de base B.

Exemples :

B :



A'_1 : "Placer au centre du repère de B le nombre d'étoiles correspondant à la valeur de la composante du vecteur des valeurs V ".

A'_2 : "Placer au centre du repère de B le chiffre correspondant à la valeur de la composante du vecteur de valeurs V ".

V : (2)

objet vide {B, A'_1, Ø}

objet {B, A'_1, V}

objet vide {B, A'_2, Ø}

objet {B, A'_2, V}

Figure 3.1

Remarque : L'ambiguïté existante au niveau des dessins représentatifs des deux objets vides précédents, est levée par le donnée des triplets {B, A'_1, ∅} et {B, A'_2, ∅}.

c) Objets à bases dynamiques :

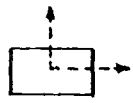
On note $A'' = C_A \quad A' = A - A'$
Soit un couple quelconque (A", V) appartenant à l'ensemble produit $A'' \times V$, et un dessin de base B quelconque appartenant à B.

(11) On appelle objet à base dynamique un dessin obtenu en appliquant la transformation T(A", V) à un dessin de base B (le dessin de base se déforme sous l'effet de la transformation T(A", V) afin de créer l'objet).

$A'' \in A''$, $B \in B \xrightarrow{T(A'', V)}$ {B, A", V} $\in$ ensemble des objets à bases dynamiques.

Un objet vide {B, A", ∅} ne possède dans ce cas aucune représentation (car le vecteur de valeur contribue directement à la création de l'objet). Cependant, afin de rester cohérent avec les définitions précédentes, nous considérons que cet objet vide est représenté par un dessin vide.

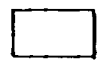
Exemples :

B : 

V : (x)

A" : "Si V = ∅ alors B = dessin vide
sinon longueur (B) = f(x) ".

Si par exemple la fonction f consiste à diviser la longueur du dessin de base par x, on obtient :

si x = 2 objet {B, A", V} : 

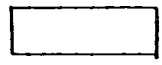
si x = 1 objet {B, A", V} : 

Figure 3.2

. notations :

T' = {t = T(A', V), A' $\in A'$, V $\in V$}

T'' = {t = T(A'', V), A'' $\in A''$, V $\in V$}

O = {ensemble des objets}

O' = {ensemble des objets à bases statiques}

O'' = {ensemble des objets à bases dynamiques}

T = $T' \cup T''$

A = $A' \cup A''$

O = $O' \cup O''$

Soit B $\in B$

$$B \xrightarrow{t' \in T'} O' \in O'$$

$$B \xrightarrow{t'' \in T''} O'' \in O$$

3/1.1.3. Les figures

a) Définitions préliminaires :

(1) On appelle dessin de liaison un dessin permettant de composer graphiquement des dessins de type objet, ce dessin de liaison pouvant être représenté par le dessin vide.

(2) On appelle algorithme de construction un algorithme permettant de construire un dessin par composition de dessins de type objet avec un dessin de liaison, la construction étant conditionnée par le nombre et le type des objets. Cette composition peut être effectuée de manière répétitive ou non suivant le type de l'algorithme considéré (le nombre de répétitions dépendant alors du nombre d'objets à composer).

. notations :

L = {ensemble des dessins de liaisons}

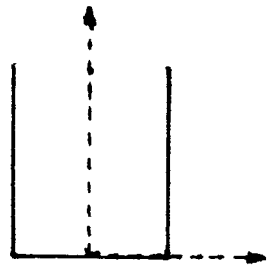
C = {ensemble des algorithmes de construction}

$P(O)$ = {ensemble des parties de O}

Soit un couple quelconque (L, O) appartenant à l'ensemble produit $L \times P(O)$, et un algorithme de construction quelconque C appartenant à l'ensemble C.

exemples:

L:



C': "empiler les objets de l'ensemble O à partir du centre du repère de L ".

$O = \{O_1, O_2, \ldots O_n\}$

avec $O_i = \{B, A'_1, V_i\}$ (objets définis à la figure 3.1.)

- si $V = \emptyset$:

aucun vecteur V_i, donc aucun objet O_i, soit $= \emptyset$

figure vide $\{L, \emptyset, C'\}$:

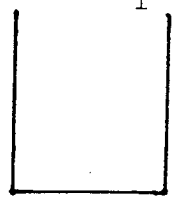


figure 3.3.

- si $V = \{V_1, V_2\}$:

avec $V_1 = (2)$ et $V_2 = (4)$

on obtient deux objets: $\{B, A'_1, V_1\}$ et $\{B, A'_1, V_2\}$

soit la figure $\{L, O, C'\} = \{L, \{B, A'_1, V_1\}, \{B, A'_1, V_2\}, C'\}$:

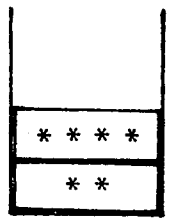


figure 3.4

(3) On appelle <u>figure</u> un dessin obtenu en appliquant l'algorithme C au couple (L, O). On note {L, O, C} cette figure.

$(L, O) \in L \times P(O) \xrightarrow{C} \{L, O, C\} \in$ ensemble des figures

(4) On appelle <u>figure vide</u> une figure construite à partir d'un ensemble d'objets vide.

$(L, \emptyset) \in L \times P(O) \xrightarrow{C} \{L, \emptyset, C\}$

(5) L'opération qui permet de passer de la figure vide à la figure telle qu'elle a été définie précédemment est appelée <u>actualisation</u> de figure.

$\{L, \emptyset, C\} \xrightarrow{\text{"actualisation"}} \{L, O, C\}$

b) <u>Figures à liaisons statiques</u> :

Soit C' l'ensemble des algorithmes de construction qui s'exécutent de manière non répétitive. La composition des objets avec le dessin de liaison est effectuée en un seul pas, et ne dépend pas du nombre d'objets à traiter. Le dessin de liaison joue alors le rôle de "support" pour le construction.

$C' \subset C$

Soit un couple quelconque (L, O) appartenant à l'ensemble produit $L \times P(O)$, et un algorithme de construction quelconque C' appartenant à l'ensemble C'.

(6) On appelle figure à liaisons statiques un dessin obtenu en appliquant l'algorithme C' au couple (L, O). (Les liaisons entre les objets qui composent la figure sont fixes et prédéfinies pendant toute la durée de vie de la figure).

$C' \in C'$, $(L, O) \in L \times P(O) \xrightarrow{C'} \{L, O, C'\} \in$ ensemble des figures à liaisons statiques

<u>Remarque</u> : Une figure vide est donc représentée dans ce cas par le dessin de liaison L.

c) figures à liaisons dynamiques

On note $C'' = C_C\ C' = C - C'$

Soit un couple quelconque (L, O) appartenant à l'ensemble produit $L \times P(O)$, et un algorithme de construction quelconque C" appartenant à l'ensemble C''.

(7) On appelle figure à liaisons dynamiques un dessin obtenu en appliquant l'algorithme C" au couple (L, O). (Les liaisons entre les objets qui composent la figure évoluent en fonction des objets de l'ensemble O).

$C'' \in L''$, $(L, O) \in L \times P(\ ) \xrightarrow{C''} \{L, O, C''\} \in$ ensemble des figures à liaisons dynamiques

Une figure vide {L, Ø, C"} ne possède dans ce cas aucune représentation (car le dessin est le résultat d'une composition entre les objets et le dessin de liaison). Nous considérons que cette figure vide est alors représentée par un dessin vide.

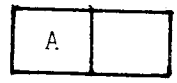
exemple:

L : 

C" : "répéter pour i=1 jusqu'à n-1:
établir un chaînage linéaire entre les objets O_i et O_{i+1}
à l'aide du dessin de liaison L ".

$O = \{O_1, O_2 \ldots O_n\}$

avec $O_i = \{B, A', ("A")\}$:



soit $V = \{V_1, V_2, V_3\}$

avec $V_1 = ("A")$, $V_2 = ("B")$ et $V_3 = ("C")$

On obtient trois objets: $\{B, A', V_1\}$, $\{B, A', V_2\}$ et $\{B, A', V_3\}$

soit la figure $\{L, O, C''\} = \{L, \{B,A',V_1\},\{B,A',V_2\},\{B,A',V_3\}, C''\}$

figure 3.5.

Notations

F = {ensemble des figures}
F' = {ensemble des figures à liaisons statiques}
F'' = {ensemble des figures à liaisons dynamiques}
$F = F' \cup F''$
$C = C' \cup C''$

Soit un couple $(L, O) \in L \times P(O)$

$$(L, O) \xrightarrow{C' \in C'} f' \in F'$$

$$(L, O) \xrightarrow{C'' \in C''} f'' \in F''$$

3/1.1.4. Les images

Les images représentent la liste des figures apparaissant sur l'écran à un instant donné (voir paragraphes suivants).

3/1.2. La sémantique graphique

3/1.2.1. Les contraintes

Seuls les objets ont accés aux "valeurs" et sont capables de se déplacer, mais aucun mouvement n'est décrit dans l'absolu. L'utilisateur ne manipule que des positions relatives d'objets par rapport à d'autres objets d'une même figure et les déplacements s'effectuent à "l'intérieur" de cette dernière.

La première contrainte découle directement de la définition même de l'objet. En effet l'objet, de par sa nature, est l'unité de base (du point de vue graphique) accessible à l'aide du logiciel graphique. Il correspond donc à l'unité de représentation d'une donnée ou d'un groupe de données, ce qui implique un accés direct aux "valeurs" représentatives des données à visualiser. Par contre, la figure représente une collection d'objets liés entre eux par des liens logiques. Elle correspond en fait à une primitive de

structuration du dessin, cette structuration étant en général le reflet de l'agencement interne du groupe de données à visualiser (tableaux, listes...). Il découle de ceci qu'une figure ne peut avoir accés aux "valeurs" que par les objets qui lui appartiennent. Cette remarque s'applique de la même manière aux images qui représentent un niveau de structuration supplémentaire. La principale différence résidant alors dans la nature temporelle de l'image par opposition à la nature permanente de la figure (du point de vue de la description graphique).

La deuxième contrainte, relative aux mouvements, comporte deux aspects d'origines différentes. Tout d'abord, le fait que seuls les objets soient capables de se déplacer découle comme précédemment directement des définitions. Par contre, la suppression des déplacements dans l'absolu provient de la nature même du système et des options prises lors de la constructions des images. Il est clair que l'emploi de déplacements dans l'absolu implique d'une part la description du mouvement, accompagnée des calculs relatifs aux déplacements, et d'autre part la connaissance exacte de la position et de la taille de chaque constituant de l'image à l'instant donné. Ceci signifie que l'utilisateur doit à nouveau gérer totalement son écran, ce qui est contraire au but fixé. Dans l'optique d'un système d'interprétation graphique il semble plus judicieux de n'introduire que des déplacements relatifs entre plusieurs objets, l'interprétation du mouvement étant conditionné par les facteurs suivants:

- la (les) valeur(s) affectée(s) à l'objet,
- le calcul de déplacement relatif effectué à l'aide d'un algorithme propre à l'objet,
- le calcul de positionnement à "l'intérieur" de la figure à laquelle l'objet appartient (position relative par rapport aux autres objets de la figure). Ce calcul est effectué à l'aide d'un tableau de correspondance (passage d'indications entre la figure et l'objet) ou d'un algorithme décrit dans la figure (clé du mouvement).

Nous citerons l'exemple d'une représentation graphique d'un index de tableau. L'objet "index" possède dans sa définition, outre un dessin de base, un

algorithme de déplacement de ce dessin en fonction de la "valeur" fournie par le système. Le fonctionnement réel de ce dessin est en fait lié aux objets de la figure à laquelle se rattache cet index. Nous remarquerons que dans ce cas là, la clé du mouvement n'est pas un algorithme (positionnement et non mouvement) mais un ensemble de valeurs numériques (bornes des tableaux) fourni par une table de correspondance, accompagné d'un formule de calcul.

Notons enfin qu'une figure est matérialisée sur la console graphique par une portion rectangulaire d'écran dans laquelle évolueront les différents composants de cette dernière. Les problèmes posés par l'allocation de ces portions d'écran aux différentes figures ont été examinés lors de l'étude de la gestion graphique.

3/1.2.2. Le comportement des objets, des figures et des images

- Les objets: Les objets statiques sont constitués d'un dessin de base fixe et d'un symbole variable qui représente la (les) valeur(s) à un instant donné.

Les objets dynamiques évoluent des deux manières suivantes:

- déformation du dessin de base dans le repère de l'objet en fonction de la(des) valeur(s) associée(s).
- déplacement du dessin de base dans le repère de la figure qui contient la "clé du mouvement".

- Les figures: Elles évoluent à l'intérieur des zones d'écran qui leur sont affectées. Dans le cas des liaisons statiques, seul le nombre ou la forme des objets qui la composent peuvent varier.

- Les images: Elles sont constituées de l'ensemble des figures et des objets qui apparaissent à un instant donné sur l'écran. Comme nous l'avons vu précédemment, la composition de ces images dépend directement du programme d'application.

3/1.2.3. La durée de vie des objets, des figures, des images

- Les objets d'une figure sont "détruits" lorsque la liste de ces objets change ou lorsque la figure disparait.

- Les figures sont "détruites" lorsque la liste des figures qui composent une image change ou lorsque l'on change de séquence (voir paragraphe animation).

- Les images sont "détruites" de la même manière que les figures (voir paragraphe animation).

3/1.2.4. L'enchaînement dynamique des images (animation)

- l'horloge :

Le temps est simulé car nous nous trouvons dans un contexte de multiprogrammation. L'enchaînement dynamique des images peut alors être rythmé par l'utilisateur à l'aide d'une touche de fonction ou son équivalent.

- animation interne:

On appelle animation interne une déformation ou un déplacement d'objet dû à la nature même de celui-ci (la description de l'objet comporte un algorithme de transformation. Un objet de base statique que l'on valorise (apparition d'un symbole ou d'un dessin supplémentaire) ou un objet à base dynamique qui se positionne provoque une animation interne. Ce niveau d'animation n'est pas connu de l'utilisateur, mais est provoqué par le système lors des mises à jour d'objets existants.

- animation externe:

On appelle animation externe toute déformation au niveau de l'unité logique que représente la figure. Le fait, par exemple, de rajouter ou de supprimer un objet dans une figure constitue une animation externe. Ce niveau d'animation est directement provoqué par le programme d'application, à chaque modification des composants de l'image.

- séquence:

On appelle séquence une liste de figures. On dit que l'on change de séquence lorsque cette liste est modifiée.

- mode d'enchaînement des images:

Comme nous l'avons vu précédemment, le programme d'application possède un certain contrôle sur l'enchaînement dynamique des images. En effet, c'est le programme lui-même qui indique à l'aide d'un ordre "afficher" lorsque l'image suivante est complète (du point de vue des données à représenter). Le rôle du système est alors d'analyser cette nouvelle image afin de décider s'il est nécessaire de changer de séquence ou autrement dit de reconstruire entièrement l'image. Dans le cas où la liste de figures n'a pas changé par rapport à l'image précédente (même séquence), le système se contente de mettre à jour les objets (animation interne) ou s'il y a lieu, de rajouter ou supprimer un ou plusieurs objets dans une figure (animation externe). Cette dernière situation peut être rencontrée par exemple lors de la visualisation d'une pile ou d'une liste.

3.2.- LE SYSTEME D'INTERPRETATION GRAPHIQUE DE DONNEES DU POINT DE VUE DE L'UTILISATEUR

3/2.1. Le mode d'utilisation du système

Dans un premier temps, l'utilisateur remplace les ordres "write" de son programme par des ordres "écran" qui possèdent quelques paramètres supplémentaires tels que le numéro de descripteur ou le blocage de données. Il rajoute ensuite un ordre d'affichage chaque fois qu'il considère qu'une image est complète (voir annexe 1§4). Dans le cas où les "descripteurs" standard conviennent, le travail est terminé. Si par contre, l'utilisateur désire "essayer" plusieurs modes de représentations de ses données, afin de décider laquelle d'entre elles est la plus satisfaisante, il change ses descripteurs à l'aide des moyens d'interaction du système (sans pour cela stopper l'exécution de son programme).

Dans le cas le plus défavorable, (aucune représentation standard ne convient), l'utilisateur doit définir lui-même son mode de représentation et pour cela déclarer de nouveaux éléments qu'il utilisera ensuite dans ses descripteurs.

3/2.2. Les données

3/2.2.1. L'aspect des données à visualiser

Comme nous l'avons vu précédemment, les données à visualiser sont fournies au système par l'intermédiaire d'un ordre de visualisation graphique, cet ordre permettant en outre d'établir une certaine structuration de ces données. Le fait que cette possibilité supplémentaire de structuration soit du plus bas niveau (concaténation) provient directement des deux remarques suivantes:

- un ordre de sortie graphique, tout au moins dans le contexte dans lequel nous nous plaçons, doit avoir une syntaxe proche de celle d'un ordre de sortie imprimante ;

- l'intégration d'un nouvel outil dans un langage de programmation ne doit pas introduire des possibilités d'un niveau totalement différent. Ainsi nous éviterons, par exemple, d'introduire des outils de structuration de données de type ALGOL 68 dans un langage aussi pauvre que FORTRAN dans ce domaine.

Si nous nous limitons volontairement dans l'apport de nouvelles primitives de structuration en ce qui concerne l'interprétation graphique des données, il n'en va pas de même pour ce qui est de l'emploi de celles existantes. En effet, le système se doit alors d'utiliser au maximum les possibilités du langage considéré. Cependant, cette remarque nous conduit directement à remettre en cause le problèmes de l'indépendance vis-à-vis du langage de programmation. Nous sommes donc amenés à prendre non pas la position la plus faible, mais une position intermédiaire qui suppose une certaine interprétation, dans le cas des langages de programmation les plus pauvres (absence de structure de liste en FORTRAN par exemple) et une sous-exploitation des possibilités des langages les plus riches.

Un autre aspect concerne la validité momentanée de certaines valeurs vis-à-vis de la représentation. En effet, un ensemble de données structurées peuvent avoir à un instant donné qu'un sens partiel du point de vue de la représentation. Il est donc nécessaire dans ce cas là, afin de clarifier l'image, de ne représenter que les données de l'ensemble qui possèdent une signification à l'instant considéré. Ce choix pouvant être effectué de deux manières différentes:

- impossibilité de représentation de la donnée (valeur infinie ou ensemble de données vide par exemple) ;
- ou spécification à partir du programme d'application.

La manipulation d'une donnée pour la représentation peut alors susciter l'utilisation des trois catégories de noms suivants:

- le nom générique qui est le nom d'une classe d'éléments,
- le nom symbolique graphique qui est le nom d'un élément graphique délcaré dans un descripteur (la portée de ce nom étant égale à la "longueur" de la séquence correspondante),
- le nom symbolique de la structure de donnée qui est le nom affecté à cet ensemble de données par le programmeur.

3/2.2.2. Le transfert des données et la structuration

3/2.2.2.1. les variables, les constantes et les expressions:

La méthode la plus simple consiste à transférer les données avec leurs structurations propres (variables simples, tableaux ...). La structure, le nombre et la valeur des données étant alors indiquées implicitement par le nom symbolique de la variable associée.

exemple:

```
begin
  integer I,J ;
  integer array A (1::5) ;
  real array B (1::2, 1::10) ;
       :
  ecran (no, I, J, I+J, true, A, B) ;
       :
end.
```

Mais ce mode de transfert devient rapidement insuffisant, car il est parfois nécessaire de visualiser des sous-ensembles de données structurées (cas par exemple d'un tableau possédant un grand nombre d'éléments). Nous sommes donc conduits à introduire l'emploi de "sous-ensembles", soit de "sous-tableaux" dans le cas des tableaux de valeurs.

exemple:

```
begin
  integer N ;
  integer array A (1::4, 1::6) ;
       :
  N:=4 ;
  ecran (no, A(1,*), A(4,1:2), A(3:N, 5:N+1)) ;
       :
end.
```

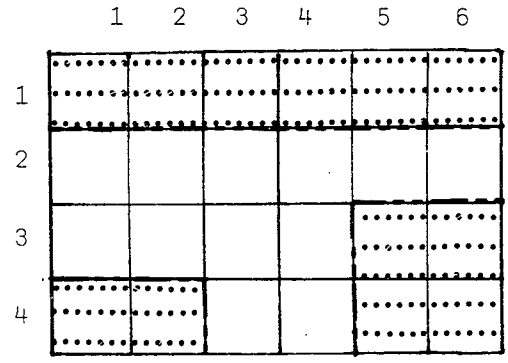


figure 3.6.: exemples de sous-tableaux

Remarque:

Nous considérons que la représentation d'un ensemble vide (sous-tableaux sans éléments par exemple) n'est pas un cas d'erreur, mais que le nombre d'objets générés est nul dans ce cas. Cette option, qui est cohérente avec les définitions, permet par exemple de visualiser une pile sans se soucier du fait que cette dernière soit vide ou non.

exemple:

```
begin
  integer array Pile (1::10) ;
  integer tete ;
     :
  ecran (no, Pile (1:tete)) ;
     :
end.
```

3/2.2.2.2. le groupement explicite de données:

Comme nous l'avons déjà vu, un ensemble de données non structurées au niveau du programme d'application, peut requérir une représentation "globale". La première possibilité consiste alors à grouper une suite de données entre des parenthèses dans un ordre "écran". L'autre méthode, qui permet de construire

des blocs de données structurées, utilise des ordres "bloc" et "finbloc" paramétrés ou non.

. exemple 1:

```
begin
  integer array X, Y(1::10) ;
     :
  ecran (no, /courbe/(X,Y)) ;
     :
end.
```

Les tableaux "X" et "Y" sont groupés linéairement pour la représentation, et "courbe" représente le nom symbolique associé au bloc de données.

Remarque:

En l'absence de spécification d'un nom symbolique, c'est le nom de la dernière variable rencontrée dans le groupement qui identifie le bloc de données.

. exemple 2:

```
begin
  string(8)S ; integer I, N ;
     :
  read(N) ;
  bloc ;
  for k:=1 until N do
  begin
    "traitement qui donne le nom et la valeur associée dans 'S' et 'I' ";
          :
    ecran (no, S, I) ;
          :
  end ;
  fin bloc (1:2, 1:N) ;
     :
end.
```

N couples (S,I) sont groupés pour la représentation. Le type de blocage de ces données est indiqué par les paramètres de l'ordre "finbloc" (En l'absence de ces paramètres, le blocage est linéaire). Dans le cas précédent, nous obtenons un tableau de dimension deux à représenter, soit par exemple:

N = 3

	1	2
1	"ID1"	1
2	"ID2"	4
3	"ID3"	0

Le choix de la représentation dépendant ensuite de la description externe (cf § 3.2.3.).

3/2.2.2.3. les données relatives à une image:

Les données transférées par l'ordre "écran", sont accumulées par le système jusqu'à la demande d'affichage (ordre "affiche"). Comme nous l'avons vu précédemment, l'image est reconstruite chaque fois que la liste de figures change, ce qui implique de fournir à chaque fois toutes les données relatives à une image. D'autre part, le fait de rajouter une donnée à l'ensemble relatif à l'image précédente, peut entraîner la reconstruction complète de l'image. Nous avons donc introduit un ordre "séquence" dont l'emploi n'est pas obligatoire, mais qui permet d'améliorer les échanges et la gestion de l'image. En effet, cet ordre permet d'indiquer au système que tout ce qui est interne à un "bloc séquence" constitue une même image, et les seules opérations possibles sont donc celles de mise à jour ou d'adjonction de dessin.

. exemple 1:

```
begin
  integer I, J, k ;
     :
  ecran (no, I) ; affiche ;
  ecran (no, I, J) ; affiche ;
  ecran (no, I, J, k) ; affiche ;
     :
end.
```

Nous avons donc dans ce cas là, trois images successives différentes, et donc obligation de référencer à chaque fois les données précédentes afin d'incorporer leurs représentations à la nouvelle image.

. exemple 2:

```
begin
  integer I, J, k ;
        :
        :
        :
 ┌sequence ;
 │ecran (no, I) ; affiche ;
 │ecran (no, J) ; affiche ;
 │ecran (no, k) ; affiche ;
 └fin sequence ;
        :
        :
        :
end.
```

Nous avons dans ce cas là une seule image à laquelle se rajoutent successivement les représentations de "J" et de "k".

La liste des facilités précédentes dont l'emploi n'est pas obligatoire, n'est qu'indicative. En effet, ces possibilités dépendent essentiellement du prétraitement du programme d'application, et non du système d'interprétation graphique. Ceci permet donc d'imaginer des outils de structuration de données totalement différents, la mise en forme étant effectuée par l'intermédiaire de la précompilation (cf § 4.3.).

3/2.3. Les descripteurs et le langage de description

Les descripteurs, qui sont stockés dans la "bibliothèque de descripteurs", sont par nature destinés à décrire des représentations d'une manière souple et non permanente. Ils sont de ce fait accessibles et modifiables à tout moment. En ce qui concerne la description même, nous distinguerons la déclaration des unités graphiques (à l'aide du langage de description),

de la création de l'objet ou de la figure qui n'est effective que lors de l'établissement du lien entre la déclaration graphique et l'ensemble de données à visualiser (cf §3.1.1.).

Les descripteurs permettent donc d'associer à une donnée ou un groupe de données structurées, une représentation graphique. Cette représentation étant constituée d'un ou plusieurs "sous-dessins types" stockés dans la bibliothèque d'éléments.

3/2.3.1. Le langage de description graphique

3/2.3.1.1. l'aspect général du langage:

Le langage de description graphique, tel que nous l'employons ici, doit essentiellement permettre de créer des dessins par l'intermédiaire de primitives évoluées. Ces primitives, qui ne sont autres que des sous-dessins paramétrés, constituent une base sur laquelle doit s'appuyer le langage de description graphique. Nous retrouvons ici les deux niveaux introduits lors de l'étude de structure de ce type de système, soit:

- La définition des représentations de base effectuée au niveau d'une console graphique virtuelle à l'aide d'un langage de programmation et d'un ensemble d'ordres graphiques élémentaires. Cet ensemble constitue le niveau le plus bas de la description, et n'est donc pas accessible directement par l'utilisateur. Il représente en fait les outils de production de dessin du logiciel graphique.

- L'emploi des représentations de base par l'intermédiaire d'un langage de niveau supérieur: le langage de description graphique.

Les représentations de base, qui sont constituées de sous-dessins paramétrés, sont rangées dans la bibliothèque d'éléments sous forme de modules exécutables.

La paramétrisation de ces sous-dessins est fonction d'une part de la forme d'utilisation dans le langage de description graphique, et d'autre part des

données fournies par le programme d'application. Le type des paramètres relatifs à la forme d'utilisation, permet de définir deux catégories de sous-dessins:

- Le sous-dessin primitif qui est paramétré uniquement par des valeurs numériques et des symboles. Il représente un type objet et ne possède de ce fait aucune structuration interne.

- Le sous-dessin structuré qui est paramétré par des valeurs numériques, des symboles et des références à des sous-dessins simples. Il représente un type figure et possède une structure interne propre au type de figure considérée. La structuration se situant au niveau des sous-dessins primitifs à partir desquels le sous-dessin structuré est construit.

Ces différentes remarques nous conduisent à introduire dans un premier temps, un langage de type "langage de commande", permettant de produire à volonté des sous-dessins paramétrés à l'aide d'un ensemble de mots clés. Cette approche devient vite insuffisante lorsque apparaissent les problèmes de structuration externe (liens logiques établis au niveau de l'utilisation des sous-dessins) et d'extensibilité. Nous avons alors choisi d'employer un langage permettant de déclarer et de créer des objets structurés (objets et figures au sens du paragraphe 3.1.), à l'aide d'un ensemble extensible de mots clés (voir annexe 4). Ces mots clés font référence aux sous-dessins paramétrés précédents, la paramétrisation étant déduite de la chaîne syntaxique correspondante.

· exemple: cell(E,5) : crée un objet destiné à représenter une valeur numérique fournie par le programme d'application. "cell" représente le mot clé permettant de référencer la classe de sous-dessins primitifs. La chaîne syntaxique "(E,5)" permet de déduire les paramètres "E" et "5" qui contribuent à définir la forme du dessin.

vecteur(cell(E,5)): crée une figure destinée à représenter un ensemble de valeurs numériques fournies par le programme d'application. Le sous-dessin structuré référencié par le mot-clé "Vecteur" est construit à l'aide du sous-dessin primitif désigné par le mot clé "cell".

Les définitions précédentes conduisent à effectuer les extensions aussi bien au niveau des mots clés que de la syntaxe et de la sémantique. Ainsi l'enrichissement de la bibliothèque d'éléments (création d'un nouveau type de sous-dessin) est accompagné de l'adjonction d'une nouvelle règle de grammaire comportant un certain nombre de fonctions sémantiques (cf § 4.2.).

3/2.3.1.2. l'emploi d'identificateurs:

Lors de l'établissement du formalisme graphique, nous avons introduit le "déplacement relatif" des objets. Cette notion de relatif suppose l'existence de liens logiques entre divers objets, ces liens ne pouvant être établis que lors de la description graphique (le programme d'application n'intervenant pas sur le mode de représentation). La première solution qui vient à l'esprit consiste à associer un nom aux sous-dessins que l'on crée (nom symbolique graphique), ce nom permettant de définir simplement les liens logiques entre sous-dessins.

Cependant, la sémantique liée à l'emploi de ces identificateurs, revêt un aspect plus complexe du fait de la nature dynamique du langage de description graphique. Nous dirons donc, dans un premier temps, que la portée d'un identificateur est directement liée à la durée de vie du sous-dessin associé.

. exemple:

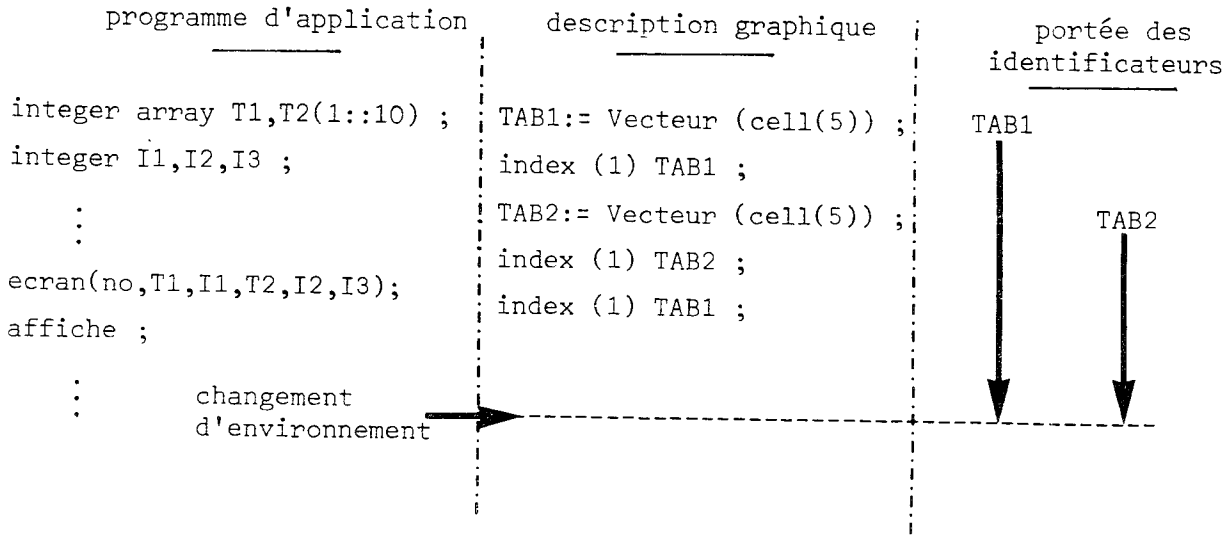
```
TAB1 := Vecteur(cell(5)) ;
TAB2 := Vecteur(cell(5)) ;
index(1)  TAB1 ;
index(1)  TAB2 ;
index(1)  TAB1 ;
```

Cette description permet de créer, en premier lieu, deux sous-dessins structurés de type "vecteur" identifiés par les noms symboliques graphiques "TAB1" et "TAB2". La création des sous-dessins primitifs de type "index", qui sont des objets dynamiques, est subordonnée à l'existance d'une figure mère (cf § 3.1.2.). Le lien logique entre l'objet et la figure étant établi, dans la mesure du possible, à l'aide des identificateurs. Dans le cas qui nous préoccupe, il sera généré trois sous-dessins primitifs de type "index" successivement dans la première figure, la deuxième puis à nouveau la première.

L'association de nom ne peut cependant être entièrement définie lors de l'analyse syntaxique et de la production de la sémantique statique. En effet, le langage permet de créer des objets structurés ou non, mais ne fournit aucun renseignement sur la durée de vie de ces derniers. La suppression d'un sous-dessin ne dépendant que de l'environnement au niveau du programme d'application et du système (cf § 3.1.2.). Ainsi une description graphique peut être correcte du point de vue de la syntaxe et de la sémantique statique et entraîner une erreur lors de l'exécution (cas par exemple de l'association de nom avec un objet ayant disparu). Il incombe donc à la sémantique dynamique d'achever l'association de nom et de détecter les erreurs dues à la nature dynamique des concepts manipulés.

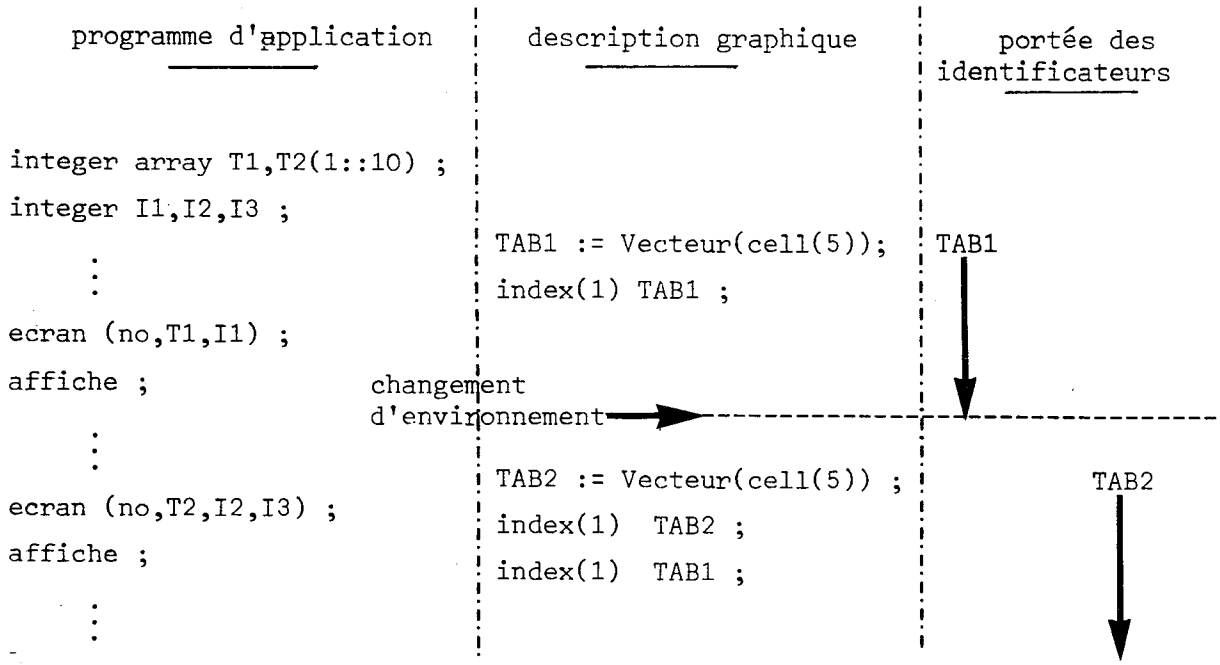
exemple:

. 1er cas:



La description graphique est correcte statiquement et dynamiquement du fait que le changement d'environnement ne se produit que plus tard. L'image produite est alors constituée de la représentation de deux tableaux, le premier étant indexé par "I1" et "I3" et le deuxième par "I2".

. 2ème cas:



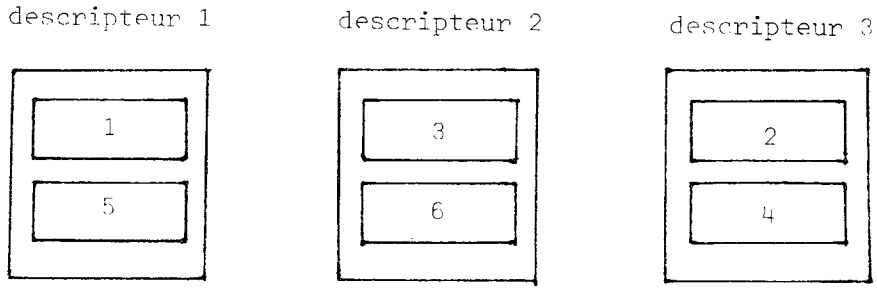
La description est correcte statiquement (syntaxe et sématique statique), mais incorrecte dynamiquement. Ceci provient du fait que le changement d'environnement est intervenu en cours des générations relatives à la description graphique précédente. Or le changement d'environnement provoque une reconstruction complète de l'image (cf § 3.1.2.4.), donc la destruction des sous-dessins existants. Ainsi, lors de la génération du deuxième index relatif au premier tableau, l'association de nom est impossible du fait de la disparition de la représentation de ce tableau et du nom symbolique graphique correspondant.

3/2.3.1.3. la modularité des descriptions graphiques

Une autre caractéristique importante de ce langage réside dans la nature modulaire des représentations graphiques employées. Il est en effet indispensable de pouvoir utiliser telle ou telle description indépendamment l'une de l'autre, et ceci parfois d'une manière quasi simultanée.

Cette utilisation simultanée de plusieurs "descripteurs" sera appelée "pseudo-parallélisme". En effet, chaque descripteur impliqué est "exécuté" tour à tour d'une manière séquentielle par morceaux (le contrôle passant alternativement de l'un à l'autre).

· exemple:



L'interprétation graphique qui produit une image à un instant t, est faite successivement à partir du descripteur numéro un, puis du descripteur numéro trois...

figure 3.7.

Ce mode d'utilisation pose alors le problème de l'accessibilité des identificateurs d'un descripteur à un autre. La première idée qui vient à l'esprit est de considérer un descripteur comme un ensemble fermé, interdisant ainsi tout lien logique entre objets créés à partir d'unités de description différentes. Cependant, le mode d'utilisation de base de ce langage (descripteurs

unitaires standard) met rapidement en évidence de graves lacunes. En effet, l'étanchéité existante entre les descripteurs oblige à traiter de nombreux cas unitaires tels que:

- représentation d'un tableau,
- représentation d'un tableau suivie d'un index,
- représentation d'un tableau suivie de deux index,

:

L'approche totalement opposée, consistant à considérer tous les identificateurs comme globaux, comporte de la même manière de nombreux inconvénients. Nous avons donc choisi une méthode intermédiaire, permettant de supprimer les inconvénients des deux approches précédentes et de n'en conserver que les avantages. Ainsi, par l'introduction de noms externes il devient possible de rendre des identificateurs globaux pour un ensemble donné de descripteurs. L'emploi de cette nouvelle facilité permet de réduire d'une manière notable l'ensemble des cas unitaires à traiter. Par exemple, tous les cas de représentation d'un tableau indexé par un ou plusieurs index, peuvent être traités à l'aide des deux descripteurs unitaires suivants:

```
descripteur 1: TAB := Vecteur (cell(5)) ; fin.
descripteur 2: externe(1) TAB ; index(1) TAB ; fin .
```

Il suffit dans ce cas d'interpréter le tableau de valeurs à l'aide du descripteur numéro un, et les divers index à l'aide du descripteur numéro deux.

Un autre type d'ambiguïté peut alors survenir en ce qui concerne l'association de nom, cette ambiguïté étant levée par le biais de la sémantique dynamique. En effet, un descripteur pouvant être utilisé plusieurs fois, peut provoquer la génération de plusieurs sous-dessins comportant le même nom symbolique graphique. Nous dirons alors que seul le dernier en date est accessible (considération uniquement dynamique).

. exemple:

programme d'application	description graphique
integer array T1, T2 (1::10) integer I ; ⋮ ecran (1,T1) ; ecran (1, T2) ; ecran (2, I) ; affiche ; ⋮	descripteur 1 : TAB := Vecteur (celle(5)) ; fin. descripteur 2 : externe(1) TAB ; index(1) TAB ; fin.

Les deux emplois successifs du descripteur numéro un permettent de générer les représentations des tableaux "T1" et "T2". L'index "I" est attaché ensuite à la représentation du tableau "T2", la représentation du tableau "T1". étant devenue innaccessible.

3/2.3.2. L'emploi des descripteurs

Nous distinguerons deux approches différentes en ce qui concerne l'emploi des descripteurs:

- L'emploi des descripteurs standard unitaires qui permet à l'utilisateur d'obtenir des représentations standard pour ses données sans avoir à préciser de description graphique. Cette approche conduit cependant à introduire un nombre plus ou moins important d'ordres "écran" unitaires dans le programme d'application (voir exemple ci-dessous). Il est toutefois possible, dans ce cas, de revenir à une situation plus saine en laissant le soin au précompilateur "d'éclater" les ordres de sorties graphiques. Ceci suppose évidemment que ce dernier soit capable de reconnaître le type des données afin d'adopter la représentation qui convient. Cette approche peut alors être considérée comme le parallèle des sorties imprimantes standard non formatées (impression standard en ALGOLW d'un entier, d'un réel, d'un logique, d'une chaîne de caractères...)

- L'emploi de descripteurs mieux adaptés au programme d'application, la description étant faite à l'aide de représentations standard ou non (voir exemple ci-dessous). Cette approche permet d'avoir une meilleure gestion des sorties graphiques, tout en permettant un certain contrôle sur les représentations choisies.

. exemple:

1/ utilisation des descripteurs standard unitaires précédents (représentation d'un tableau: (1), et d'un index:(2)):

```
begin
real array P,T(1::10) ; integer I,J ;
  .
  .
  .
ecran (1,T) ;                    ┐
ecran (2,I) ;                    │  affichage du tableau T indexé par I
affiche ;                        ┘
  .
  .
  .
ecran (1,T) ;                    ┐
ecran (2,I) ;                    │
ecran (2,J) ;                    │  affichage du tableau T indexé par I et J,
ecran (1,P) ;                    │  et du tableau P indexé par I+J
ecran (2,I+J) ;                  │
affiche ;                        ┘
  .
  .
  .
end.
```

. remarque: Dans l'optique précédente d'une représentation des données effectuée d'une manière totalement standard (fonction du type des données), nous ne rencontrerions que deux ordres "écran" référençant un numéro de descripteur fictif (0 par exemple): écran(0,T,I) et écran (0,T,I,J,P,I+J).

La restitution de la configuration précédente étant alors effectuée par le précompilateur avec parfois une interprétation différente (une valeur entière à la place d'un index par exemple).

2/ utilisation d'un descripteur "utilisateur":

```
begin
real array P,T(1::10) ; integer I,J ;
  :
ecran(3,T,I) ;                    ] affichage du tableau T indexé par I
affiche ;                         ]
  :
ecran(3,T,I,J,P,I+J) ;            ] affichage du tableau T indexé par I et J
affiche ;                         ] et du tableau P indexé par I+J
  :
end.
```

Descripteur 3:

```
A:= vecteur(cell(5)); index(1)A ;
B:= vecteur(cell(5)); index(1)B; index(1)B;
C:= vecteur(cell(5)); index(1)C ;
```

} dans la bibliothèque de descripteurs

fin.

3/2.4. L'interaction au niveau de l'image

L'emploi de descripteurs standard conduit à automatiser, dans une certaine mesure, la gestion de l'écran. Il est en effet impossible, dans ce cas, d'inclure des renseignements sur la taille et la position des éléments à l'intérieur de ces descripteurs utilisables à partir de n'importe quel programme d'application. Une gestion complète de l'image peut être d'une part lourde et coûteuse et d'autre part inadaptée aux besoins réels d'un utilisateur donné. Nous avons donc choisi d'employer une méthode mixte qui a l'avantage

de laisser le choix à l'utilisateur, tout en fournissant (à la demande) des configurations possibles. Ainsi la gestion automatique de l'écran n'est faite que lorsque les descriptions ne contiennent aucune indication sur la taille et la position des éléments (cas des descripteurs standard). Mais là encore l'utilisateur peut imposer sa propre configuration à l'aide des actions cataloguées de première catégorie (détection d'un sous-dessin, application d'une affinité ou d'une translation). Si on se place dans l'optique d'une réalisation destinée à avoir des rééditions futures, il devient indispensable de pouvoir enregistrer les diverses actions produites par l'utilisateur afin de restituter à chaque image la configuration désirée. Ceci implique la création d'un fichier utilisable parallélement à celui des descripteurs mais avec une contrainte supplémentaire: il est strictement lié au programme d'application. Il parait alors plus judicieux de construire des descripteurs avec des indications de taille et de position dans ces cas là. Nous remarquons toutefois que la construction de ces descripteurs peut être effectuée de deux manières différentes:

- soit directement par l'utilisateur,
- soit automatiquement par le système, lors de la première édition du "film" (en tenant compte des diverses actions effectuées par l'utilisateur).

La deuxième catégorie d'actions cataloguées permet de modifier le mode de représentation, donc de remplacer certains sous-dessins. Ces actions agissent en fait au niveau des descripteurs ou de la définition des éléments de base. Un certain nombre d'outils graphiques est alors mis à la disposition de l'utilisateur (édition graphique du fichier contenant les descripteurs, modifications de ces descripteurs...).

3/2.5. L'interaction au niveau de l'enchaînement dynamique des images

La troisième catégorie d'actions cataloguées permet de modifier l'enchaînement dynamique des images (suppression ou rétablissement d'un certain nombre d'images). Ceci suppose que l'utilisateur est capable de se souvenir de la numérotation des ordres "afficher" dans le programme d'application, afin de désigner ceux qui sont impliqués dans l'action en cours. Dans le cas contraire, il serait souhaitable que, dans une version plus évoluée, le programme

source puisse être affiché sur l'écran, afin que l'utilisateur "désigne" à l'aide d'un outil graphique (photostyle, réticule) les ordres concernés.

Deux possibilités sont offertes en ce qui concerne la numérotation des ordres "afficher":

- laisser le travail au précompilateur (numérotation séquentielle),
- définir des niveaux de visualisation par le biais d'une numérotation adéquate.

L'utilisateur peut par exemple affecter un numéro de son choix à tous les ordres de visualisation inclus dans une boucle englobée, afin de pouvoir stopper à un moment donné la visualisation de ce niveau de détail.

. exemple:

```
boucle 1
|   :
|  affiche(1)
|  boucle2
|  |   :
|  |  affiche(2)
|  |   :
|  |  affiche(2)
|  |   :
|  fin
|   :
|  affiche(1)
|   :
fin.
```

On est en présence dans ce cas de deux niveaux de visualisation, le détail

de la boucle intérieure pouvant être annihilé par désactivation (actions cataloguées de troisième catégorie) du deuxième niveau de visualisation.

3/2.6. La déclaration des nouveaux éléments

3/2.6.1. Aspect général

Le système possède un ensemble d'éléments de base qui est stocké dans la "bibliothèque d'éléments". Cet ensemble, ainsi que nous l'avons déjà vu, peut être enrichi par des dessins-types paramétrés propres à l'utilisateur, et cela à l'aide d'un langage de haut niveau, et d'un ensemble de facilités dites graphiques (outils évolués de production de dessins, outils de structuration...). Ces mécanismes qui permettent de définir de nouveaux éléments, doivent sous-entendre d'une part la nouvelle syntaxe (cf § 3/2.3.1.) et d'autre part, le mode de production du nouveau dessin-type en fonction des paramètres de description (forme de référence de cet élément dans un descripteur) et des données à visualiser. Nous trouverons donc essentiellement, en ce qui concerne l'aspect description, les trois catégories d'outils suivantes:

- les outils de production de dessin qui permettent aussi bien d'obtenir directement une figure géométrique (rectangle, cercle, ellipse...), que de définir une liste de coordonnées (les segments étant jointifs ou non) ou encore une chaîne de caractères.

- les outils de structuration qui permettent "d'attacher" des objets (anciens ou nouveaux sous-dessins de type objet) à des figures.

- les outils d'introduction de valeurs qui en fait permettent d'obtenir le type et la valeur de la donnée à visualiser courante. Nous classerons également dans cette catégorie, les outils qui permettent d'obtenir des renseignements en vue de la définition d'un mouvement quelconque. Nous citerons par exemple l'obtention de la valeur des bornes de la figure à laquelle est attaché un index en cours de production.

Le formalisme relatif aux déclarations de nouveaux éléments étant traité au paragraphe 4.2., nous ne traiterons dans ce paragraphe, que l'aspect purement extérieur.

Ainsi une déclaration d'un nouveau type "objet" se présente sous la forme générale suivante:

```
classe objet <nom de classe objet> <liste de paramètres formels>
begin
|     dessin := ... ;
|     contraintes := ... ;
|     nom graphique := ... ;
end ;
```

La liste desparamètres formels étant dans ce cas constituée uniquement de variables simples (pas de structuration possible à ce niveau).

Une déclaration d'un nouveau type "figure" se présente sous une forme analogue:

```
classe figure <nom de classe figure>  <liste de paramètres formels>
begin
|     dessin :=... ;
|     algorithme de "génération" d'objets dans la figure, structuration ...
|     nom graphique := ... ;
end ;
```

Toutefois, dans cette catégorie d'éléments, la liste de paramètres formels peut contenir, outre des variables simples, des références à des éléments de type "objet". C'est donc essentiellement à ce niveau que la structuration est effectuée à l'aide des primitives de structuration et des paramètres formels qui représentent des références à des objets.

Enfin, au niveau du système, chaque déclaration d'une nouvelle "classe" donne naissance d'une part à une extension de la grammaire (cf § 4.2.), et d'autre part, à un module exécutable rangé dans la bibliothèque d'éléments.

3/2.6.2. Les outils employés

Nous ne donnerons pas dans ce paragraphe une liste définitive et complète des outils disponibles pour définir les extensions, mais un certain nombre d'exemples dans chaque catégorie. En particulier, en ce qui concerne les outils de production de dessin, il existe un très grand nombre de possibilités aussi bien dans les techniques purement graphiques (tablette sylvania par exemple) que dans les techniques informatiques numériques (génération de figures géométriques telles que des polygones ou des coniques...).

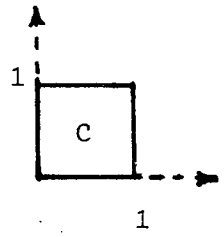
a) Parmi les outils de production de dessin, nous citerons:

- la production de polygones et de coniques à l'aide de fonctions,
- la production d'un dessin à l'aide d'un dispositif graphique (tablette sylvania par exemple),
- la production d'un dessin à l'aide d'une description sous forme de liste de couples de coordonnées. Les atomes collatéraux de cette liste étant considérés comme des points joints par des segments de droites ou des chaînes de caractères suivant le cas.

. exemple:

```
dessin := ((0,1) (0,0) (1,0) (1,1) (0,1)) (0.5,0.5,"C") ;
```

La variable graphique "Dessin" contient le dessin suivant:



b) En ce qui concerne la structuration, nous nous limiterons dans un premier temps à l'emploi d'une fonction qui permet de construire une figure à l'aide d'objets "formels" ou "effectifs".

c) Le dernier type d'outil s'adresse en fait à la paramétrisation des sous-dessins par l'apport de valeurs. Nous distinguerons alors les

outils permettant d'introduire des valeurs provenant directement du programme d'application (données à visualiser), des outils dont le rôle est de fournir des renseignements sur des sous dessins structurés déjà créés. Notons enfin, que les outils de première catégorie peuvent aussi bien fournir les valeurs des données à visualiser que des renseignements sur le type, le nombre ou encore la forme de ces données.

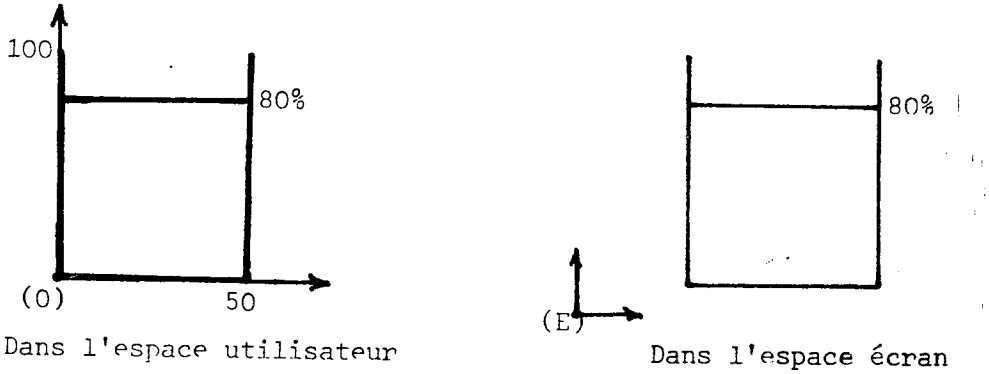
3/2.6.3. exemples d'extensions

Nous rappellerons que dans les exemples qui suivent, le choix du repère pour la description des dessins n'intervient pas dans la représentation finale. Ceci provient du fait que cette description est effectuée dans l'espace utilisateur et que le passage à l'espace écran est effectué par le système en fonction des divers critères vus précédemment.

1) Définition d'une pile, la donnée à visualiser indiquant le taux d'occupation:

```
classe objet pileniv
begin type objet dynamique deformable ;
    |   real xx ; string (3) S ;
    |   xx := donnée ;
    |   S := codage (xx) ;
    |   S(3,1) := "%" ;
    |   dessin := ((0,100)(0,0)(50,0)(50,100)) ((0,xx)(50,xx)(50,xx,S)) ;
end.
```

La représentation d'une donnée de "valeur 80" est alors la suivante:



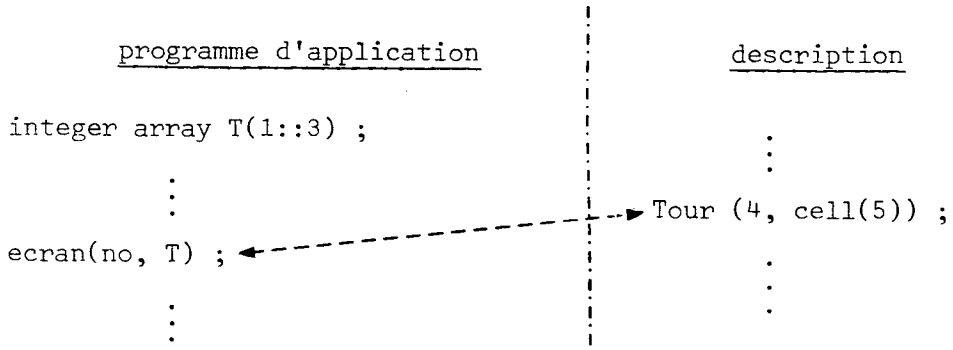
Dans l'espace utilisateur Dans l'espace écran

figure 3.8

2) Définition d'une "tour" dans laquelle on empile des objets:

```
classe figure Tour (N, objet ØB)
begin  type figure statique ;
       real deltax, deltay, x, y ;
       integer nb ;
       x := y := 0 ;
       nb := nbdonnée ;
       for k := 1 until nb do
       begin  attache (ØB, x, y, deltax, deltay) ;
              y := y + deltay ;
       end ;
       y := deltay * N ;
       dessin := ((0,0)(deltax,0)) ((deltax/2,0)(deltax/2,y)) ;
end.
```

Ce nouveau type de représentation peut être ensuite utilisé avec des objets statiques standard par exemple:



La représentation est alors la suivante (pour des valeurs de 30, 20 et 10):

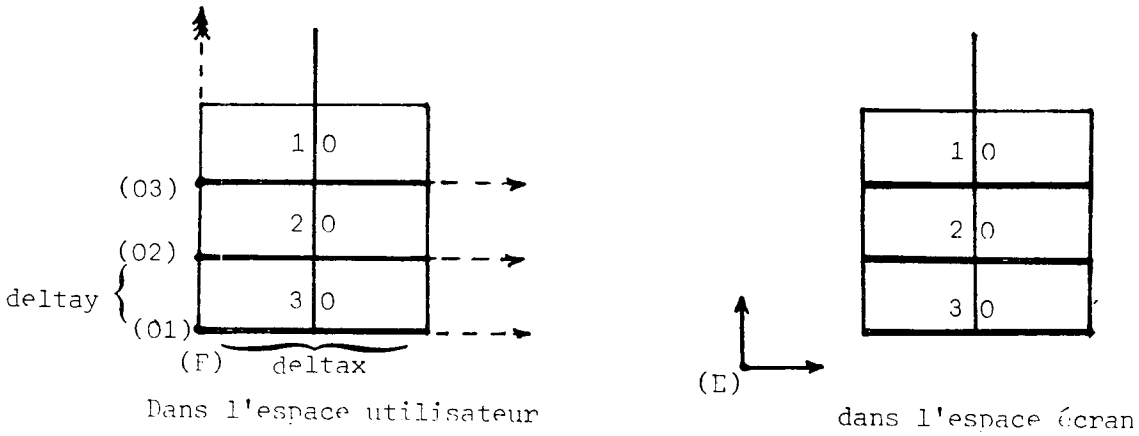
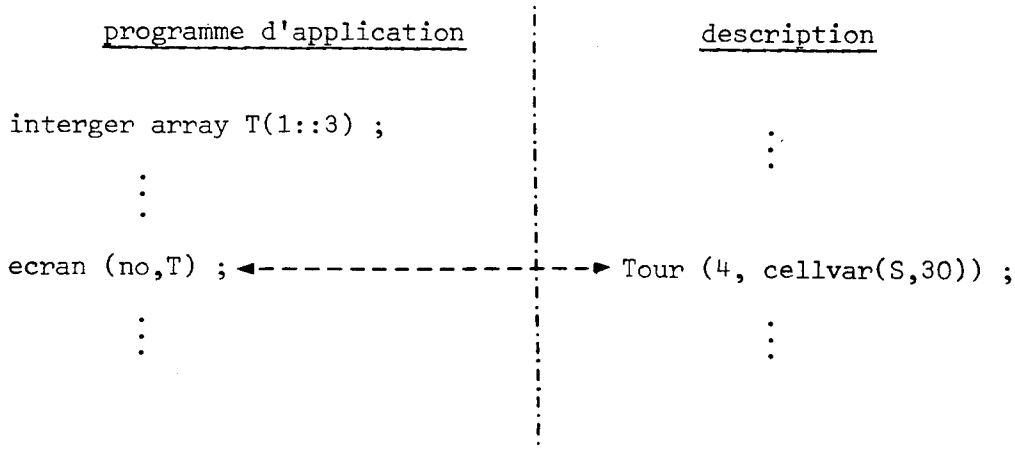


figure 3.9

De même, ce nouveau type de représentation peut être utilisé de la manière suivante (voir exemple des "Tour de Hanoï"):



La représentation est alors la suivante (pour des valeurs par exemple de 30, 20 et 10):

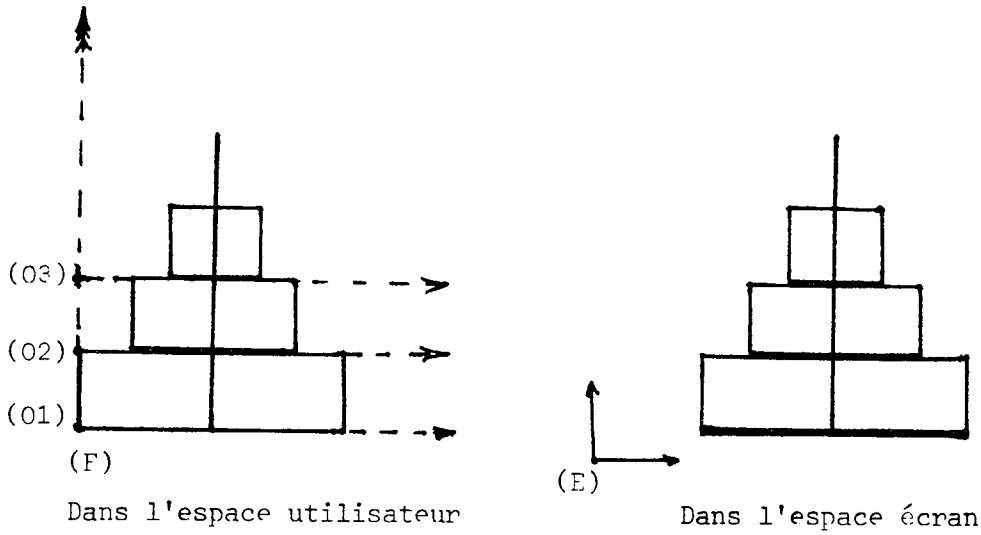


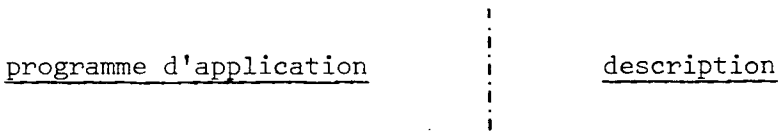
figure 3.10

3) Définition d'une figure "horloge" dans laquelle les objets "aiguille" tournent en fonction des données à représenter.

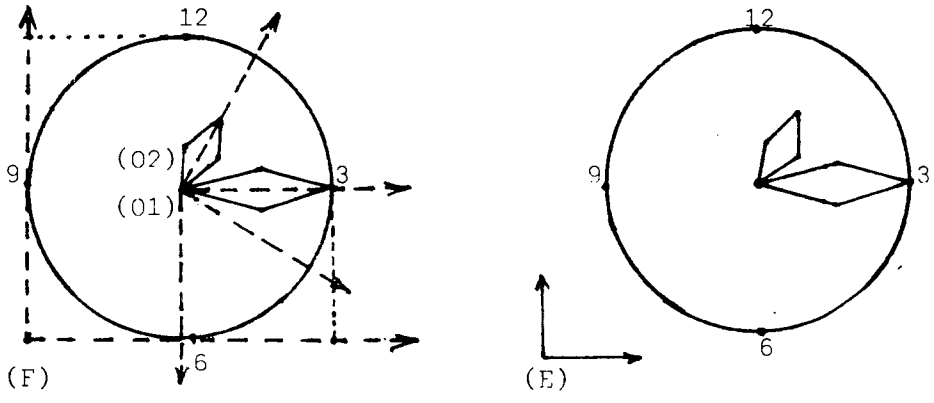
```
classe figure horloge
begin type figure statique ;
      real deltax, deltay ;
      dessin := cercle (2)+(1,2,"12")(2,1,"3")(1,0,"6")(0,1,"9") ;
      attache (aiguille(1), 1, 1, deltax, deltay) ;
      attache (aiguille(2), 1, 1, deltax, deltay) ;
end.

classe objet aiguille (N)
begin type objet dynamique mobile ;
      real teta, x ;
      x := donnée ;
      dessin := ((0,0)(0.25,N/2)(0,N)(-0.25,N/2)(0,0)) ;
      if N=1 then teta := (3.14/12) * x else teta := (3.14/60) *  x ;
      mouvement (0, 0, teta) ;
end.
```

Nous remarquerons que la définition de la figure "horloge" n'est pas "générale" (pas de paramétrisation de type objet et pas de variation possible sur le nombre d'objets). En effet, cette figure est définie à l'aide d'un dessin de base fixe et de deux objets dynamiques bien définis. Les deux objets dynamiques qui représentent les aiguilles de l'horloge, sont attachés au centre le la figure, puis subissent une rotation appliquée par la fonction "mouvement".



La représentation du couple (1,15) est alors la suivante:



Dans l'espace utilisateur Dans l'espace écran

figure 3.11

Enfin, en ce qui concerne les erreurs d'utilisation, un certain nombre de vérifications sont effectuées par le système. Il est toutefois évident qu'il incombe à l'utilisateur de prendre un certain nombre de précautions lors de la définition de nouveaux éléments, le système n'étant pas sensé connaître les particularités des diverses extensions. Ainsi, si le système est capable de vérifier la concordance des types entre les données à visualiser et leur mode d'utilisation lors de la définition de l'extension, il n'en est pas de même en ce qui concerne la forme d'utilisation de certains paramètres. Nous citerons par exemple l'objet "aiguille" dont le paramètre "N" ne doit prendre que les valeurs un ou deux.

Nous terminerons enfin en précisant que le système se charge d'assurer la compatibilité, dans la mesure du possible, entre le type de la donnée à visualiser et le mode de représentation, et cela en effectuant au besoin certaines conversions.

3/2.7. Exemple de correspondance (définition-déclaration-création):

a) Définition d'un "type objet" (nouveau type de représentation):

```
classe objet cell (I1, I2)
begin
  :
end.
```

Comme nous l'avons vu, nous obtenons ainsi une extension au niveau de la grammaire, accompagnée de la production d'un module exécutable étiqueté par "cell" dans la bibliothèque d'éléments.

b) Déclaration d'un élément graphique (dans un descripteur):

```
A := cell(4,2) ;
```

c) Création de l'élément graphique:

C'est l'appel correspondant effectué à partir du programme d'application qui fait le lien entre la donnée (écran(...x...)) et la déclaration de l'élément graphique.

Le module exécutable étiqueté par "cell" dans la bibliothèque d'éléments est activé. Les paramètres effectifs sont alors les suivants:

- l'entier 4,
- l'entier 2,
- le réel x fourni par le programme d'application,
- le nom "x" fourni par le programme d'application.

Remarque :

"cell" est le nom générique, "A" est le nom symbolique graphique et "x" est le nom symbolique de la donnée.

Chapitre 4

LE CHOIX DES MECANISMES GENERAUX ET LEURS IMPLEMENTATIONS

4.1.- LE TRAITEMENT DU LANGAGE DE DESCRIPTION GRAPHIQUE

La nature particulière du langage de description graphique implique un traitement qui, bien que s'apparentant à des techniques conventionnelles, possède parfois un aspect plus ou moins original.

4/1.1. Le choix du type d'analyse syntaxique

En premier lieu, nous devons choisir entre une méthode "descendante" et une méthode"ascendante".

Les méthodes descendantes [L3] n'ont besoin pour s'exprimer que des règles de la grammaire, du fait que partant de "l'axiome" elles essaient par des "dérivations" successives, d'obtenir la chaîne du texte à analyser. La dénomination descendante provenant de la conception "descendante" de l'arbre syntaxique associé au texte analysé (de la racine vers les feuilles qui représentent le texte analysé).

Les méthodes ascendantes (simple précédence [L4], précédence faible [L5]... ont besoin pour leur part d'un type d'information différent: les relations de précédence. En effet, il ne s'agit plus de construire des dérivations mais au contraire de reconnaître des parties droites de règles de grammaire afin de "réduire" la chaîne traitée. Ainsi, dans une analyse de ce type, on doit retrouver l'axiome après des "réductions" successives du texte source.

Le choix d'une méthode descendante, qui parait de prime abord arbitraire, s'explique ensuite par les diverses propriétés que l'on désire attribuer au langage à traiter. En effet, ce langage, de par sa nature même, doit être modelable. La première idée qui vient alors à l'esprit est d'utiliser une méthode classique de production d'automate de traitement syntaxique (compilateur de compilateur [L6]),le traitement du langage étant alors effectué d'une manière prédictive [L3]. Cependant, cette technique devient rapidement lourde lorsqu'il s'agit d'incorporer des modifications "mineures dans le langage considéré, ce type de modification dépendant d'un autre mécanisme: les extensions. Nous emploierons donc une méthode mixte permettar

suivant le cas, de traiter le problème avec la première ou la deuxième technique. Ainsi, les modifications qui n'altèrent pas la "base du langage" (cf § 4.2.) seront traitées par des mécanismes d'extension faciles à manipuler par l'utilisateur. Par contre, la modification "globale" du langage, qui ne s'adresse pas à un utilisateur isolé mais à une certaine implémentation collective, met en oeuvre des techniques plus complexes. En effet, le langage est alors défini par un ensemble de règles de grammaire et de règles sémantiques qui sont à la base de l'implémentation. Afin de fournir un outil capable de générer automatiquement l'implémentation d'un tel langage, nous avons adopté une technique de représentation de la grammaire sous forme de liste. Cette technique, ne pouvant évidemment traiter automatiquement tout langage (notamment en ce qui concerne la sémantique) nous avons établi un certain nombre de contraintes à respecter. En premier lieu, nous nous limitons aux langages du même type que celui introduit précédemment (cf § 3.2.3.) afin de posséder un noyau commun de règles sémantiques. Dans ces conditions, nous dirons que le langage est entièrement défini par la donnée des règles de grammaire et l'énumération des règles sémantiques employées.

D'autre part, nous supposons que la grammaire est non ambigüe, non récursive à gauche (contraintes relatives aux méthodes descendantes d'analyse syntaxique)et qu'elle se présente sous une forme LL(1). Ceci, dans le but évident d'éviter les retours arrière coûteux lors de l'analyse syntaxique, et de permettre l'inclusion des fonctions sémantiques dans n'importe quelle règle de grammaire. Cette structure permet à l'automate de travailler en un seul passage et de produire directement le code interne (cf § 4.1.3.).

La sémantique statique relative à un texte analysé est ensuite définie par l'ensemble des fonctions sémantiques référencées dans les règles de grammaire concernées par l'analyse syntaxique. La sémantique dynamique, pour sa part, est produite directement à partir de l'implémentation générale. Les règles qui la régissent sont de ce fait difficilement modifiables.

4/1.2. Représentation de la grammaire sous forme de liste avec inclusions sémantiques:

La grammaire est représentée en mémoire par une liste à deux niveaux. Le premier niveau correspond aux diverses règles avec leurs différentes alternatives (uniquement des pointeurs). Le deuxième niveau correspond aux chaînes syntaxiques de la grammaire (terminaux et non-terminaux). Nous distinguerons alors les trois catégories de terminaux suivants:

- les atomes (symboles de base de la grammaire, mots clés et élément vide) qui permettent de guider l'analyse syntaxique,
- les identificateurs qui possèdent en plus une valeur de désignation,
- et les nombres qui indiquent une valeur numérique à exploiter.

Ce découpage implique que l'analyse lexicographique reconnaisse ces divers types de terminaux à l'aide d'une grammaire de niveau inférieur (reconnaissance des identificateurs, des nombres et des symboles spéciaux), afin que chaque élément du langage soit traité correctement (établissement des correspondances à l'aide des identificateurs, association des valeurs numériques à l'aide des nombres...). Nous rajouterons toutefois au niveau du traitement, une nouvelle catégorie de terminaux qui n'appartiennent pas au vocabulaire de la grammaire (qui n'apparaissent donc pas dans le langage) mais qui sont présents dans les règles de grammaire: ce sont les fonctions sémantiques. En ce qui concerne la représentation interne, les métasymboles sont remplacés par des pointeurs dans la liste (pointeurs vers des éléments de premier niveau), les atomes par des pointeurs vers les chaînes de caractères correspondantes (zone des atomes), les identificateurs et les nombres par des symboles spéciaux et les fonctions sémantiques par leurs numéros d'identification. La grammaire est fournie au départ sous forme de BACKUS , les fonctions sémantiques et les fins de règles étant distinguées à l'aide de symboles spéciaux n'appartenant pas au vocabulaire de base de la grammaire.

Exemple: soit la grammaire suivante qui permet de reconnaître des objets et de les générer:

<inst> ::= <element> <S>

<S> ::= ; <inst> | ∅

<element> ::= ident 'f1' := figure (nombre 'f2')|<objet>⇒ ident 'f1'

<objet> ::= ob(nombre 'f2' x nombre 'f2') 'f3'

V_T = {; := figure ( ) ⇒ ob x ∅} U {ident nombre} U {'f1' 'f2' 'f3'}

V_N = {inst element S objet}

Les fonctions sémantiques f1, f2 et f3 permettent de traiter respectivement les identificateurs (association de nom), les nombres (paramétrisation) et la génération des objets paramétrés. On obtient alors la représentation interne suivante:

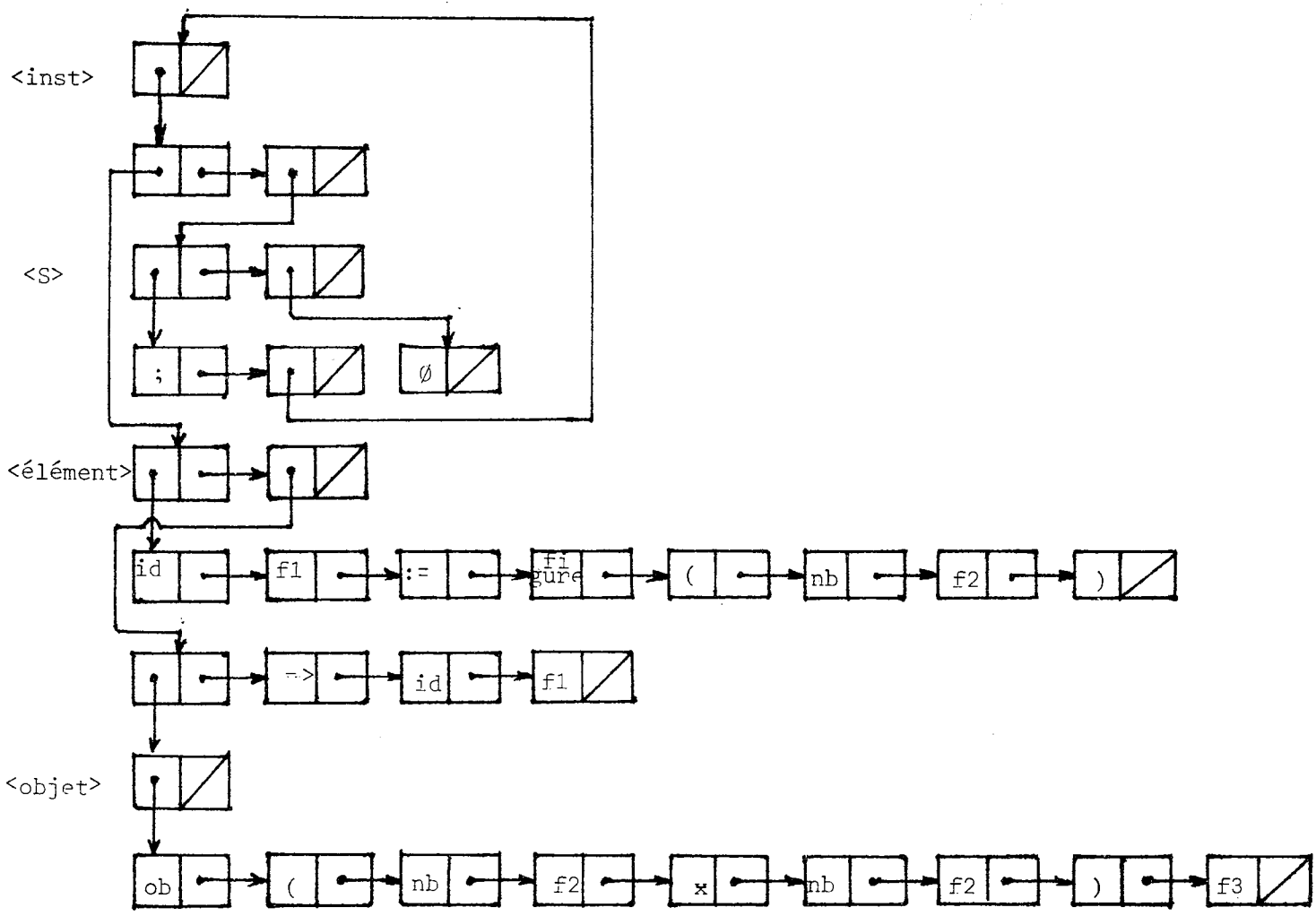


figure 4.1.

Nous remarquons que les éléments du premier niveau qui correspondent aux métasymboles, possèdent comme descendants directs les sous-listes de deuxième niveau (qui représentent les chaînes syntaxiques) et comme collatéraux les différentes éventualités.

4/1.3. L'analyse syntaxique et les productions sémantiques associées

Notons G la grammaire et L_G le langage engendré. Le fait d'analyser la syntaxe d'une phrase écrite dans le langage L_G équivaut à réaliser un parcours de la liste associée aux règles de la grammaire G, ce parcours étant guidé par les éléments terminaux qui constituent la phrase. Si la phrase est syntaxiquement correcte, le parcours effectué permet de "revenir" à l'axiome (point de départ du parcours de la liste). Ce parcours, qui est souvent récursif, est représenté par l'arbre syntaxique de la phrase, et est accompagné par les productions sémantiques associées.

Exemple: soit la phrase suivante écrite à l'aide du langage précédent:

A := figure(1) ; ØB(4 x 2) ⇒ A

Le parcours de liste correspondant est le suivant:

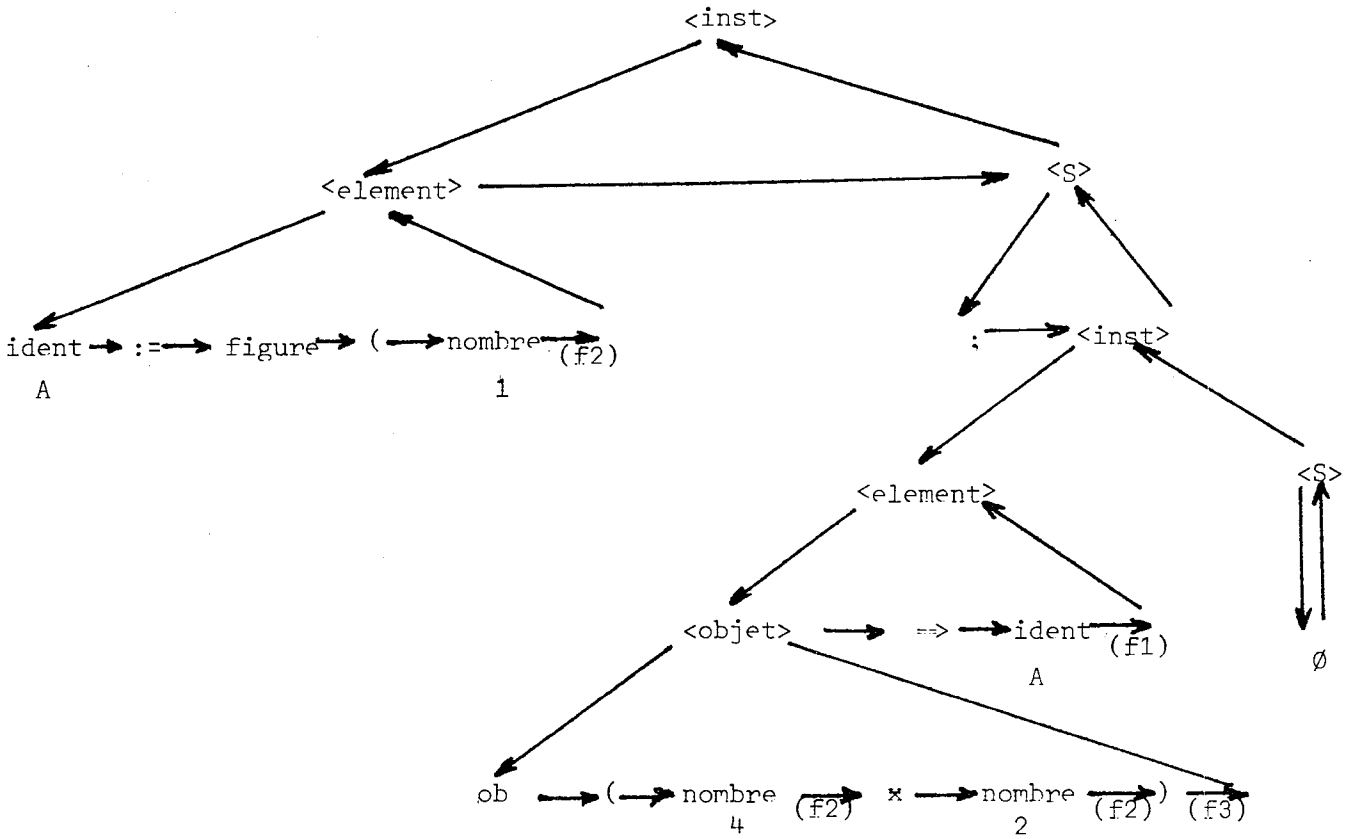


figure 4.2.

Le code généré correspondant est le suivant:

- création d'une figure paramétrée par 1,
- génération d'un objet de catégorie 'ØB' paramétré par 4 et 2, et attaché à la figure précédente.

L'algorithme de base du parcours de la liste à deux niveaux est décrit par les procédures récursives suivantes:

. notations:

"courant" correspond à l'élément à traiter dans la phrase et "suivant" à l'élément suivant.

"succés" est une variable logique globale.

"méta" est une procédure qui permet de préciser si nous sommes en présence d'un métasymbole ou non.

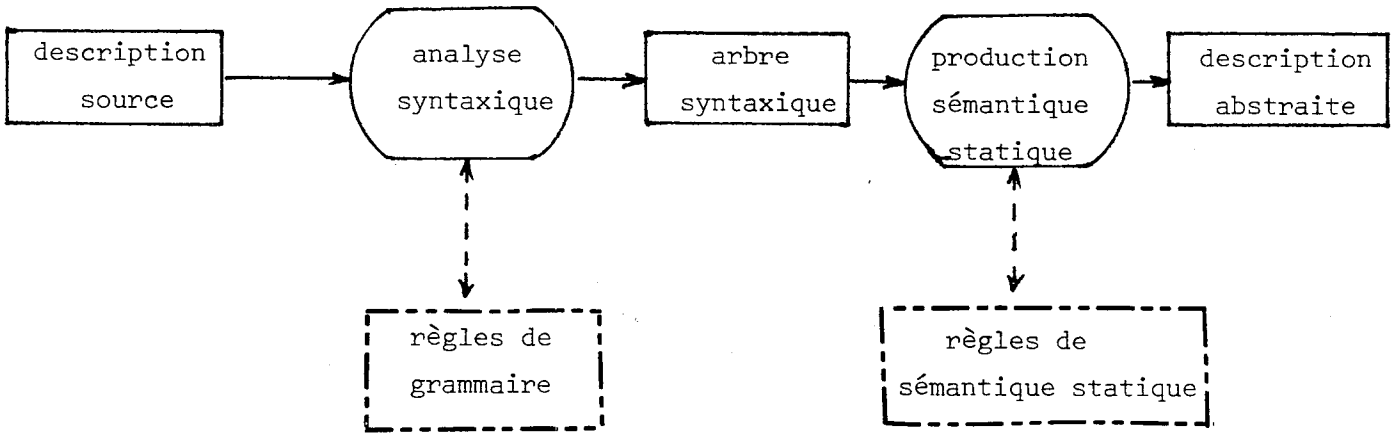
```
procédure event (I)
début succés := faux ;
|     tant que (I¬ = NIL)∧(succés) faire
|     début règle (tête(I)) ;
|     |     si ¬succés alors I := queue(I) ;
|     fin ;
fin ;
procédure règle(I)
début succés := vrai ;
|     tant que (I¬ = NIL)∧(succés) faire
|     début si meta(tête(I)) alors event (tête(I)) sinon
|     |     si tête(I)¬ = courant alors succés := faux sinon
|     |     début succés := vrai ;
|     |     |     courant := suivant ;
|     |     fin ;
|     |     I := queue(I) ;
|     |     "traitement sémantique s'il y a lieu"
|     fin ;
fin ;
```

Pour lancer l'analyse d'une phrase il suffit d'activer:

"event (tête de liste)".

4/1.4. La "description source" et la "description abstraite"

Nous appelons description source, toute description graphique effectuée à l'aide du langage L_G. Par opposition, nous définirons la description abstraite qui est une description graphique interne "sémantiquement équivalente". Le passage de la description source à la description abstraite étant effectué à l'aide des mécanismes décrits précédemment et suivant le schéma classique [E1]:

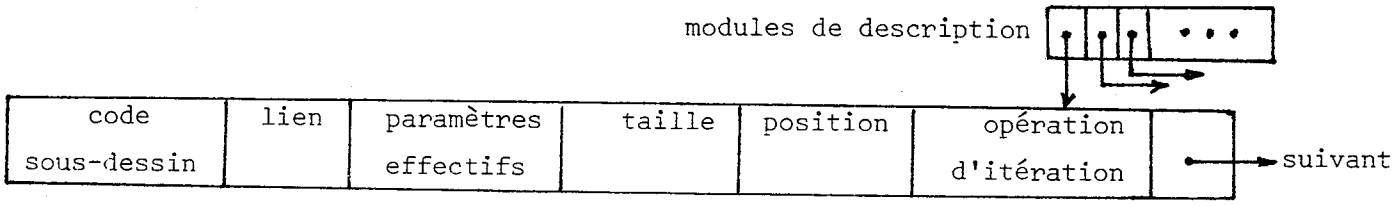


La description abstraite représente alors l'arbre syntaxique accompagné de renseignements d'origine sémantique tels que l'édition de liens logiques ou la paramétrisation statique de certaines actions.

Le nombre des catégories d'actions possibles dans un langage L_G du type défini précédemment étant limité, nous avons adopté une structuration simple au niveau interne. Cette structuration s'appuie sur le fond même du langage c'est-à-dire sur la référenciation d'objets structurés et paramétrés. Notons qu'à ce niveau la paramétrisation et la structuration des objets sont purement statiques, l'effort dynamique n'étant effectué que lors de l'interprétation de la description abstraite. Nous ne nous intéressons donc pour l'instant , qu'à la paramétrisation des sous-dessins, relative au type de représentation choisi. Ces paramètres effectifs, dont le nombre

peut varier en fonction de mode d'utilisation, sont rangés en mémoire dans une zone structurée réservée à cet effet. Cette technique permettra par la suite de générer des sous-dessins par simples appels aux modules exécutables correspondants, ces derniers se chargeant de récupérer les paramètres effectifs à l'aide d'une procédure spécialisée.

La représentation interne de la référenciation aux sous-dessins paramétrés est alors la suivante:



élément de liste (référenciation d'un sous-dessin paramétré)

En ce qui concerne la gestion de la liste des paramètres effectifs, qui peuvent comporter entre autre des références à des sous-dessins type paramétrés, nous avons adopté le formalisme suivant:

- génération de la liste des paramètres effectifs numériques relatifs au sous-dessin considéré(un paramètre omis est remplacé par l'infini).
- génération de la liste des paramètres effectifs qui sont des références à d'autres sous-dessins. Chaque référence étant suivie des paramètres effectifs de niveau inférieur relatif au sous-dessin référencé.
- génération de la liste des pointeurs vers les différentes références paramétrées.

Exemple: soient une figure et deux objets référencés respectivement par Fig,Ø1 et Ø2 avec la syntaxe suivante (un paramètre pouvant être omis est noté entre crochets):

Fig(val.numérique [,val.numérique] ; ref.objet, ref.objet)

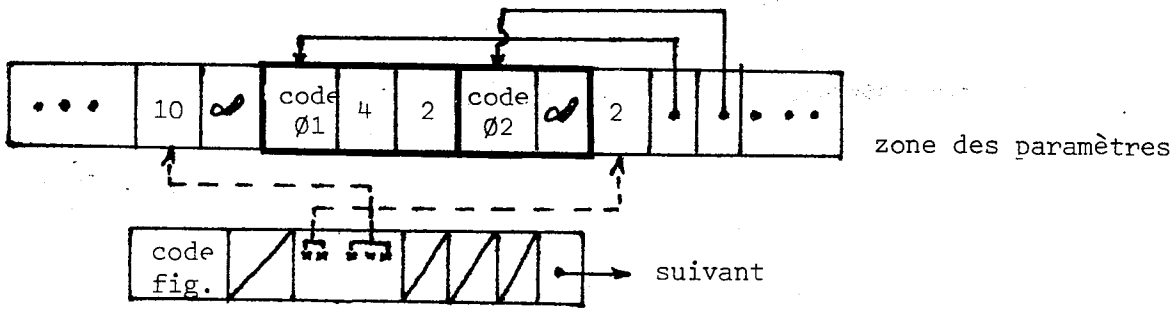
Ø1(val.numérique,val.numérique)

Ø2 [(val.numérique)]

Soit la référence suivante à la figure Fig:

Fig(10;Ø1(4,2),Ø2)

La représentation interne associée est alors la suivante:



élément correspondant de la liste de référenciation

4/1.5. L'interprétation de la description abstraite et la production des sous-dessins

La description abstraite est interprétée à l'aide d'un ensemble de routines sémantiques qui aiguillent sur telle ou telle action. C'est à ce niveau que se place la sémantique dynamique et notamment les actions suivantes:

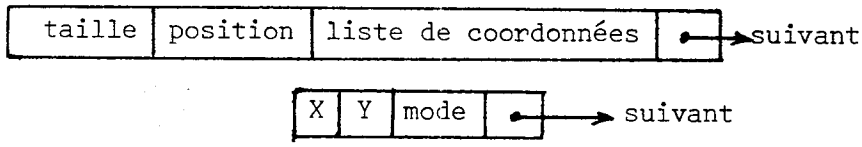
- édition des chaînages dynamiques (structuration pour le mouvemer par exemple),
- paramétrisation supplémentaire des sous-dessins (données fournie par le programme d'application),
- génération des sous-dessins paramétrés.

Cette interprétation permet de gérer une liste qui représente l'image, chaque élément de liste représentant une unité graphique. Les renseigne ment portés étant alors les suivants:

- la référence au sous-dessin paramétré correspondant,
- le chaînage de structuration dynamique,
- le bloc de données à visualiser ainsi que son nom symbolique,
- et la table de correspondance pour les mouvements.

Cette liste permettra ensuite au système de gérer la console de visualisation.

Parallélement, la génération des sous-dessins produit une liste appelée liste d'affichage. Cette liste comporte des suites de coordonnées non dimensionnées (coordonnées des objets dans l'espace utilisateur), assorties de paramètres qui indiquent la forme de tracé à adopter (segments consécutifs ou disjoints, chaînes de caractères...).



Avec cette méthode, toute translation ou déformation affine d'un sous-dessin, est définie sans modifier la liste de coordonnées. Ainsi, une mise à jour ou une modification interactive de ce type, ne demande au départ que la mise à jour de la tête de liste. Il est évident, toutefois, que si la console graphique employée n'offre aucune possibilité de mise à jour, il faudra recalculer les différentes coordonnées lors de la génération du code propre au dispositif graphique considéré (rôle de l'interface).

Nous remarquerons que jusque là, parmi les mécanismes mis en oeuvre, aucun ne dépend du matériel graphique employé. En effet, nous avons repoussé le plus loin possible tout ce qui pouvait créer un asservissement quelconque au matériel, et en particulier la génération du code propre à la console de visualisation employée.

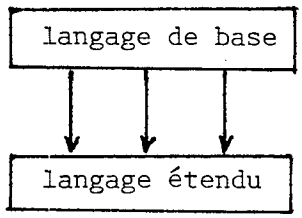
Ce n'est donc qu'en dernier lieu, que les coordonnées seront converties par le biais d'une double affinité et d'une translation (cf§2.4.).

4.2.- LES EXTENSIONS

4/2.1. L'aspect général des extensions du langage

"Dans le sens général, un langage extensible est conçu comme étant un langage de programmation qui, en plus de la possibilité d'écrire des programmes d'application, fournit divers outils pour modifier sa propre définition et, en conséquence, sa propre implémentation" [E1].

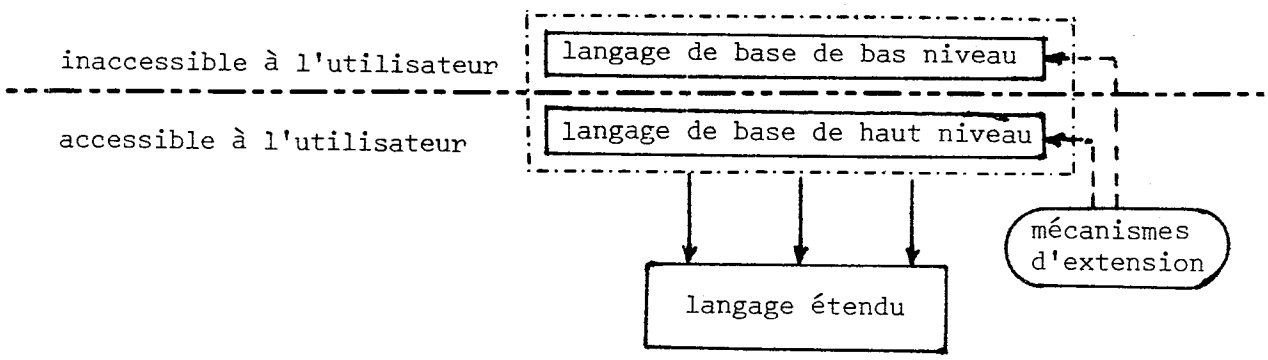
Afin de rester cohérent avec la définition des langages, nous distinguerons l'extension syntaxique de l'extension sémantique. La syntaxe permettant de spécifier les conventions à respecter pour construire chaque phrase du langage et ainsi de dégager la structure des phrases, la sémantique permettant pour sa part d'attacher aux constructions précédentes un certain nombre d'actions qui constituent la signification de ces constructions. Les moyens fournis pour modifier aussi bien la syntaxe que la sémantique sont appelés mécanismes d'extension, ces derniers s'appuyant sur le langage de base. Dans le sens habituel, les mécanismes d'extension permettent de combiner entre eux des éléments primitifs aussi bien que des éléments déjà construits, afin d'en définir des nouveaux. La définition nouvelle entraînant automatiquement l'implémentation du nouvel élément.



mécanismes d'extension

En ce qui nous concerne, cette définition parait de prime abord insuffisante. En effet, nous désirons avoir en outre des possibilités de combinaisons d'éléments existants, des moyens de créer de toute pièce de nouveaux éléments (un nouveau type de dessin peut très bien être défini sans référencer les types déjà existants). En fait, si on regarde de plus près, nous constatons que ces mécanismes font appel à des éléments de base de bas niveau, qui bien qu'inaccessibles à l'utilisateur, n'appartiennent pas moins au langage de base.

Nous distinguerons donc le langage de base de bas niveau inaccessible à l'utilisateur (cf § 3.2.3), du langage de base de haut niveau avec lequel sont constituées les phrases.



Les mécanismes d'extension permettent donc de définir simplement de nouveaux éléments. Il ne s'agit donc pas de définir des extensions par des mécanismes proches de l'implémentation d'un compilateur ou d'un interpréteur (syntaxe à l'aide de règles de grammaire, sémantique...), mais par le biais du langage même. Ceci implique donc la présence de mécanismes d'extension qui fonctionnent par l'intermédiaire d'un langage spécialisé: le langage d'extension.

Nous nous trouvons donc en présence de trois types de langages:

- le langage de base(L_B),
- le langage complémentaire qui permet l'utilisation des extensions(L_E)
- et le langage de définition des extensions(L_D).

Le langage de base étant lui-même subdivisé suivant les deux catégories suivantes:

- le langage de base de bas niveau (L_{BB}),
- et le langage de base de haut niveau (L_{BH}).

Une description graphique est donc constituée d'un texte écrit à l'aide du langage L_{BH} et L_E, ce texte étant traduit par l'intermédiaire du langage L_D en un texte écrit à l'aide du langage L_{BB}.

$$\text{Texte}_E + \text{Texte}_{BH} \xrightarrow[\text{(traduction)}]{\text{Texte}_D} \text{Texte}_{BB}$$

4/2.2. L'aspect syntaxique des extensions

Il s'agit de modifier l'aspect externe du langage en introduisant la possibilité d'utiliser des notations différentes.

Ceci suppose l'existence de moyens permettant de spécifier:

- la forme de la nouvelle notation,
- son mode d'utilisation,
- et sa signification.

Nous sommes en fait en présence d'un mécanisme de macros syntaxiques, la paramétrisation de ces macros pouvant se présenter sous deux formes différentes:

- les paramètres sont des chaînes de caractères et l'extension n'entraîne pas de modifications de la grammaire (cas par exemple des macro assembleurs IBM 360).
- les paramètres sont des catégories syntaxiques et l'extension entraîne la modification des règles de la grammaire.

Dans le cas qui nous intéresse, la deuxième forme de paramétrisation semble convenir plus particulièrement à nos besoins. L'analyse des nouvelles constructions permettant alors de guider la sémantique associée.

Exemple: classe F → new (Var1 x ref Var2)

```
begin
    .
    .   définition de l'extension à l'aide de la
    .   variable simple "Var1" et de la variable
    .   référence "Var2"
end
```

"classe F" permet de définir le type de l'extension, donc le point d'extension choisi dans la syntaxe et ainsi le mode d'utilisation du nouvel élément. La forme de la nouvelle notation est indiquée par la chaîne "new(Var1xref Var2)" qui donne naissance à une nouvelle règle de grammaire (cf § 4.2.4.). Enfin, la signification est donnée par le corps du bloc qui suit, cette signification faisant appel à la sémantique attachée aux variables impliquées.

4/2.3. L'aspect sémantique des extensions

Nous ne nous intéresserons dans ce paragraphe qu'à l'aspect "sémantique statique". En effet, la "sémantique dynamique", qui dépend directement du fonctionnement interne de l'implémentation, n'est pas accessible, tout au moins en ce qui nous concerne, à l'utilisateur. Nous dirons donc qu'il n'y a pas à proprement parler, d'extension au niveau de la sémantique dynamique.

La sémantique statique permet donc de compléter la définition des nouvelles sortes d'objets manipulables dans le langage en spécifiant la configuration de ces objets et les règles de leur utilisation.

Nous trouvons dans la thèse de JORRAND [E1] trois catégories d'extensions sémantiques:

- la définition de types,
- l'adjonction de propriétés additionnelles,
- et l'emploi des classes.

Le mécanisme des classes, qui est le plus complet, s'accorde très bien avec nos objectifs. Nous retrouvons en effet une certaine similitude entre les principes de base des classes ([E2],[E3]) et nos propres définitions. Une classe étant considérée comme un "hôte" pour éléments potentiels, et cette classe pouvant avoir un certain nombre de relations avec d'autres classes (relations d'inclusion essentiellement en ce qui nous concerne).

4/2.4. L'implémentation de l'extensibilité

Comme nous l'avons vu précédemment, le type d'extension employé modifie la syntaxe et l'implémentation du langage, et ceci par le biais du langage même et d'un ensemble de mécanismes d'extension. Toutefois, les modifications apportées sont soumises à un certain nombre de règles, notamment en ce qui concerne la syntaxe. Il est en effet prévu, lors de l'implémentation, de fournir un certain nombre de points d'application pour les extensions, et ceci à l'aide des règles de grammaire relatives au langage. Ces "points d'extension" de la grammaire sont désignés par un ensemble de métasymboles appartenant à la description grammaticale de la syntaxe du langage. Ainsi, le langage extensible sera défini syntaxiquement, pour la représentation sous forme de listes par:

- la liste des règles de grammaire sous forme de BACKUS
- la liste des métasymboles qui désignent les règles extensibles,
- et la syntaxe des extensions.

Comme nous l'avons vu précédemment, un texte écrit à l'aide d'un langage extensible est soumis à un processus de traduction qui a pour but de produire un texte "sémantiquement équivalent" exprimé à l'aide du langage de base. Dans ce qui suit, nous distinguerons le rôle de chacune des deux parties qui constituent le traducteur [E1]:

- la partie qui traite l'introduction des nouvelles extensions et dont l'effet est de modifier la définition courante du langage,
- la partie qui effectue le travail de traduction proprement dit, en se basant sur la définition courante du langage.

En fait, en ce qui concerne l'implémentation, nous nous rapprochons du traitement interprétatif illustré par la définition d'EULER [E4]. En effet, la structure syntaxique, définie par un ensemble de règles de grammaire, permet de "guider" la sémantique qui est souvent définie par l'exécution d'un ensemble de règles d'interprétation. Cependant, l'aspect rigide attaché à la définition d'EULER est atténuée dans notre cas par la production d'une description intermédiaire représentant l'arbre syntaxique du texte ("programme" de description graphique). Ceci permet donc de différer l'interprétation qui n'est d'ailleurs que partielle, et donc de détacher une partie du traitement sémantique statique et dynamique. Cette présentation syntaxique intermédiaire est ensuite complétée par un ensemble de renseignements de type sémantique (chaînages logiques, valeurs de paramétrisation), ce qui permet d'obtenir une description graphique "abstraite" (cf § 4.1.4.).

Exemple:

a/ soit l'ensemble des règles suivantes, qui permet de définir la syntaxe des extensions dans le langage de l'exemple précédent:

<extension> := <type> → <mot clé> <chaîne synt> ⊥ <def>
<type> ::= <"metal"> | <"meta2"> | ...
<mot cle> ::= ident
<chaine synt> ::= <synt><chaine synt> | ∅

<synt> ::= [nombre]<liste param>| ["meta1"]<liste param> | ...
<liste param> ::= ident <liste param>|<separ><liste param>|Ø
<separ> ::= .|,|;|...
<def> ::= "definition du nouvel element"

b/ soient les deux métasymboles suivants, qui définissent les points d'extension:
<element>
<objet>

Remarque: Dans les règles précédentes élément remplace "meta1" et objet remplace "meta2".

c/ On se donne l'exemple d'extension suivant:

<element> → newfig ([nombre]I1 : I2; [objet] Ø1, Ø2) ⊥
"définition de la nouvelle figure à l'aide des paramètres I1 et I2 de type valeur numérique, et des paramètres Ø1 et Ø2 de type référence objet"

Ceci a pour but de générer un module qui représente le nouvel élément paramétré et d'intégrer une nouvelle éventualité à la règle de grammaire désignée par le métasymbole <élément>. La nouvelle forme de la règle de grammaire (avec les inclusions sémantiques) étant alors la suivante:

<element> ::= ident 'f1' := figure(nombre 'f2')|<objet>⇒ ident 'f1'|
newfig(nombre 'f2' : nombre 'f2' ; <objet> , <objet>)

d/ Le nouvel élément ainsi défini peut alors être utilisé de la manière suivante (par exemple):
newfig(4 : 2; ØB(1 × 2), ØB(2 × 3))

L'implémentation d'un tel automate pose évidemment un certain nombre de problèmes, mais permet une utilisation plus souple du langage considéré sans pour cela alourdir outre mesure les mécanismes de reconnaissance et de traduction.

L'utilisation de ces mécanismes permet d'introduire les facilités suivantes:

- la création d'un automate, qui reconnait un langage du type précédent, est effectuée à partir de la lecture des règles de la grammaire associée accompagnée de la donnée des points d'extension ainsi que des fonctions sémantiques.

- toute modification du langage à reconnaître est entièrement définie par la modification des règles de grammaire correspondantes dans le fichier en entrée. Il devient donc possible de changer le langage sans pour cela "modifier" l'automate de reconnaissance.

- toute extension du langage entraîne l'intégration de nouvelles règles dans la grammaire (nouveaux éléments dans la liste interne) et permet ensuite une analyse simple du langage étendu (reconnaissance, détection d'erreurs, génération de code...).

4.3.- LE PRE-TRAITEMENT

4.3.1. Aspect général

Afin de compiler correctement les ordres graphiques, il est nécessaire de faire subir un prétraitement au programme d'application. Cette précompilation, qui est spécifique au langage de programmation utilisé, permet de transformer les ordres de sorties graphiques en appels de sous-programmes externes compatibles avec le langage. Chaque sous-programme permettant de transférer une unité logique de donnée (variable simple entière, tableaux d'entiers, variable simple réelle...).

Le précompilateur doit donc être capable de reconnaître les différentes variables du programme d'application, afin de générer les appels corrects des sous-programmes externes de transfert des données. Les fonctions de base du précompilateur sont alors les suivantes:

- reconnaissance des types des variables (construction d'une table des symboles),
- génération du transfert des données à l'aide des sous-programmes externes,
- structuration des données à visualiser,
- numérotation séquentielle des ordres d'affichage lorsque l'utilisateur n'a pas défini une numérotation propre (niveaux de visualisation).

Il est bien évident que ces quatre fonctions de base peuvent revêtir des aspects différents suivant le langage de programmation considéré et les options prises pour la manipulation des données. En effet, la table des symboles qui est dynamique en ALGOL W, devient statique en FORTRAN. Ou encore, une première version qui ne traite que les variables et les constantes, peut engendrer une version plus évoluée qui traite les expressions (le traitement se borne en fait à la reconnaissance du type de ces expressions, afin de produire des générations correctes).

C'est au niveau de la conception et de la structuration des données que peuvent apparaître de nombreuses variantes. En effet, pour des raisons

d'indépendance vis-à-vis de l'application et du langage de programmation employé, nous avons choisi de ne transférer que des structures simples existantes dans à peu près tous les langages de programmation. Il incombe donc à la précompilation de mettre en forme les données à visualiser.

exemple:

Soit le programme écrit en ALGOL W:

```
Begin
        integer I, J ; logical B
        real array T1, T2 (2::10) ;
        :
        ecran (1, I, J, T1(I+J)) ;
        :
        ecran (1, T1(2:I), T2, I+1) ;
        affiche(B) ;
        :
        ecran (1,(I, T1(J), T2(J+1))|A|) ;
        affiche ;
        :
end.
```

Le précompilateur produira le programme suivant:

```
Procédure Prog (procédure paramentier, tabentier, paramreel...) ;
Begin
        integer I, J ; logical B ;
        real array T1, T2(2::10) ;
        :
```

passage des données avec leurs "noms symboliques"

```
        paramentier(1, I, "I", true) ;
        paramentier(1, J, "J", true) ;
        paramreel(1, T1(I+J), "T1(I+J)", true) ;
```

```
                        .
                        .
                        .
passage des données     [ tabreel(1, T1,"T1", 2, I, true) ;
avec leurs              |
"noms symboliques"      | tabreel(1, T1, "T1", 2,10, true) ;
                        |
                        L paramentier(1, I+J, "I+J", true) ;
ordre d'affichage       [ affiche(1,B) ;
                        .
                        .
                        .
passage des données     ┌ bloc(true) ;
avec blocage            |
                        | paramentier(1,I,"I",true) ;
                        |
                        | paramreel(1,T1(J),"T1(J)", true) ;
                        |
                        | paramreel(1,T2(J+1), "A", true) ;
                        |
                        └ bloc(false) ;
ordre d'affichage       [ affiche(2,true) ;
                        .
                        .
                        .
                        end.
```

Remarque

1/ En l'absence de spécifications, le nom symbolique associé à une expression est constitué de la chaîne de caractères correspondante à l'expression.

2/ En l'absence de numérotation des ordres d'affichage, le précompilateur effectue une numérotation séquentielle afin de retrouver les ordres concernés lors d'une interaction ultérieure.

Ces précompilations, qui peuvent revêtir des aspects différents suivant les services demandés, posséderont toujours une phase de "génération" commune pour un langage de programmation donné. En effet, seule la forme des éléments à traiter en entrée peut varier. Nous distinguerons alors deux types de traitement:

- la traduction des ordres de manipulation de données,
- et l'interprétation de certains concepts relatifs au programme d'application même.

Le premier type de traitement, qui reflète la fonction de base du précompilateur, permet de mettre en forme les données pour le traitement graphique. Nous pouvons aller dans ce domaine du simple précompilateur qui traduit des ordres de sorties imprimantes, jusqu'au précompilateur évolué qui travaille à partir d'ordres et de primitives de structuration évoluées. Nous retrouvons donc toujours les deux tendances qui s'opposent: la puissance et la simplicité d'utilisation. Notre position tente d'établir un compromis entre ces deux tendances afin d'obtenir une utilisation simple, tout en offrant au besoin des possibilités supplémentaires.

Le deuxième type de traitement s'adresse à des formes d'illustration bien définies. En effet, il s'agit de demander, à l'aide d'options spéciales, la visualisation automatique de certains concepts. Nous citerons, par exemple, la visualisation du graphe du programme ou encore l'illustration de la récursivité de certains sous-programmes. En ce qui concerne la récursivité, le précompilateur doit alors être capable de générer le code nécessaire à la gestion des piles (paramètres par valeurs, variables du sous-programme, pile des appels...) et à la construction de l'arbre des appels. Il est en effet nécessaire de reconstruire ces piles pour les visualiser car le programme d'application n'a pas accès directement à la pile d'exécution (voir exemple "TOUR DE HANOI").

4/3.2. Implémentation

En premier lieu, le précompilateur doit reconnaître les déclarations et les ordres graphiques. Cette recherche qui est simple en FORTRAN (une instruction au plus par carte), demande un niveau de compréhension du programme plus élevé en ALGOL W. En effet, il est alors nécessaire de reconnaître les déclarations, les blocs, les instructions simples et la structure des instructions composées, afin de localiser les ordres graphiques. Le traitement est donc basé sur une syntaxe partielle du langage considéré.

Exemple:

Reconnaissance partielle des instructions en ALGOL W:

<inst> ::= <inst simple>|<inst composée>
<inst simple> ::= <ordre graphique>|<random>
<inst composée> ::= <inst cond>|<inst choix>|<inst iter>
<inst cond> ::= if <random> then <inst>| if <random> then <inst simple> else <inst>
<inst choix> ::= case <random> of begin <liste inst> end
<inst iter> ::= for <random> do <inst>| while <random> do <inst>
<random> ::= "saute tous les items, y compris les chaînes et les expressions conditionnelles et choix. Arrêt sur un ";" ou un des mots réservés suivants: then, else, do, of, end"

Si la syntaxe à reconnaître est représentable par une grammaire LL(1), il est possible d'utiliser la méthode exposée lors de l'analyse syntaxique du langage de description graphique. Il suffit dans ce cas, de définir les règles de grammaire et les fonctions sémantiques associées, pour obtenir l'automate recherché (automate de traitement d'un langage défini à l'aide d'une grammaire mise sous forme de liste avec inclusions sémantiques). Les règles de grammaire étant alors les règles relatives à la syntaxe partielle du langage de programmation, et les règles relatives à l'emploi des ordres graphiques. La sémantique, pour sa part, étant constituée des fonctions de gestion de la table des symboles et des routines de traitement des ordres graphiques.

4.4.- UNE METHODE D'ALLOCATION D'ESPACE-ECRAN

4/4.1. Principe de la méthode

Soit B un ensemble de blocs rectangulaires. On note B_i ou encore (h_i, l_i) le ième bloc qui possède une hauteur h_i et une longueur l_i.

La méthode consiste en premier lieu à grouper entre eux un certain nombre de blocs, afin d'obtenir des constructions de tailles peu différentes. La taille et le nombre de ces constructions permettent ensuite de calculer les dimensions de l'écran fictif, le rapport de l'affinité à appliquer et le découpage de l'écran.

4/4.1.1. Recherche de "groupements verticaux":

On appelle "groupement vertical" un ensemble de blocs $\{B_{i_1}, B_{i_2} \ldots B_{i_m}\}$ qui, pour h donné, vérifie la propriété suivante:

$$\sum_{j=1}^{m} h_{i_j} \leq h$$

Ces blocs sont alors destinés à être disposés verticalement sur l'écran.

La recherche de ces groupements peut être interprétée comme la recherche d'une partition de l'ensemble B, cette partition étant assujettie à vérifier un certain nombre de propriétés.

. Notations

$$B = \{B_1, B_2 \ldots B_n\} \text{ avec } B_i = (h_i, l_i)$$

$$h_{max} = \max(h_1, h_2 \ldots h_n)$$

$$l_{max} = \max(l_1, l_2 \ldots l_n)$$

$$T = \{\text{ensemble des applications de } B \text{ dans N}\}$$

Soit τ une application de l'ensemble T, qui vérifie les propriétés suivantes:

1/ τ est une surjection de l'ensemble B dans l'ensemble $\{1,2,\ldots m\}$

2/ l'image réciproque de $\{1\}$ ne comporte qu'un seul élément noté $\beta_1 = \{B_1^1\}$

Soit $p \in \{2,3,...m\}$

On note $\beta_p = \{B_1^p, B_2^p \ldots B_{n_p}^p\}$ l'image réciproque de $\{p\}$ par τ.

L'ensemble des images réciproques des ensembles $\{1\}$, $\{2\}$... $\{m\}$ forme une partition de l'ensemble B que l'on note: $B' = \{\beta_p, p \in 1,2 \ldots m\}$

ou encore $B' = \{(B_1^1), (B_1^2, B_2^2, \ldots B_{n_2}^2), \ldots (B_1^m, B_2^m, \ldots B_{n_m}^m)\}$

On possède donc la propriété suivante:

$\text{card}(\beta_1) + \text{card}(\beta_2) + \ldots \text{card}(\beta_m) = 1 + n_2 + n_3 + \ldots n_m = n$

On note (h_i^p, l_i^p) le couple qui représente l'élément B_i^p.

L'application τ est définie de manière à respecter les propriétés suivantes:

3/ L'élément B_1^1 est chosi de manière à avoir $h_1^1 = h_{max}$

4/ $\forall p \in 2, 3 \ldots m \quad \sum_{i=1}^{n_p} h_i^p \leq h_{max}$

5/ $\begin{cases} \forall p \in 2,3 \ldots m \\ \forall p' \in p+1, p+2 \ldots m \\ \forall i \in 1,2 \ldots n_p \\ \forall j \in 1,2 \ldots n_{p'} \end{cases}$ $\nexists B_j^{p'}$: $\begin{cases} h_j^{p'} \leq h_{max} - \sum_{k=1}^{i-1} h_k^p \\ \text{et} \\ l_j^{p'} > l_i^p \end{cases}$

Chaque ensemble de cette partition correspond à un groupement "vertical" de blocs.

exemple:

$n = 8$

$B_1 = (12, 20)$ $B_2 = (15, 30)$ $B_3 = (30, 15)$

$B_4 = (10, 10)$ $B_5 = (10, 10)$ $B_6 = (12, 5)$

$B_7 = (12, 5)$ $B_8 = (10, 8)$

$B = \{B_1, B_2 \ldots B_8\}$

τ : B → N

B_1 → 2

B_2 → 2

B_3 → 1

B_4 → 3

B_5 → 3

B_6 → 4

B_7 → 4

B_8 → 3

$$\tau(B) = \{1, 2, 3, 4\}$$

La partition correspondante de B est alors la suivante:

$$\{\beta_1, \beta_2, \beta_3, \beta_4\} = \{(B_3), (B_2, B_1), (B_4, B_5, B_8), (B_6, B_7)\}$$

Cette partition peut se traduire par le schéma suivant:

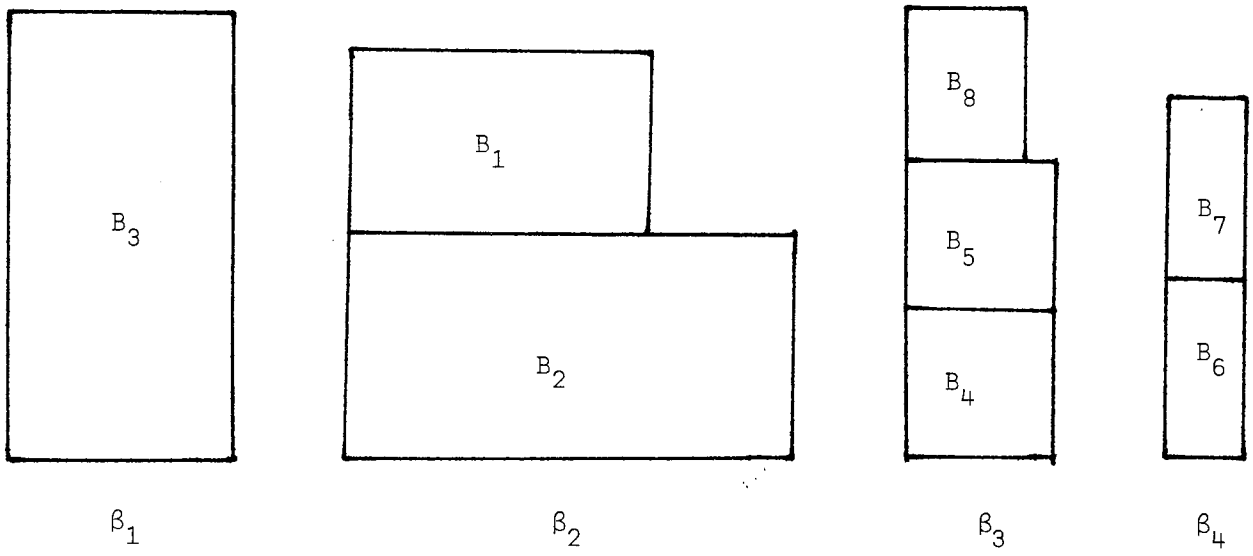


figure 4.3.

4/4.1.2. Recherche de "groupements horizontaux"

On appelle groupement horizontal un ensemble de blocs $\{B_{i_1}, B_{i_2} \ldots B_{i_m}\}$ qui pour l donné, vérifie la propriété suivante:

$$\sum_{j=1}^{m} l_{i_j} \leq l.$$

Ces blocs sont alors destinés à être disposés horizontalement sur l'écran.

Nous allons en fait rechercher des groupements "horizontaux" non pas parmi les blocs B_i, mais parmi les groupements "verticaux" précédents. Ceci revient à rechercher une nouvelle partition $\{\alpha_q, q \in 1, 2 \ldots t\}$ de B, cette partition (qui n'est pas unique) possédant la propriété suivante:

soit $\{\beta_q, p \in 1, 2 \ldots m\}$ la partition précédente de B

6/ $\forall p \in 1, 2 \ldots m, \forall q \in 1, 2 \ldots t \quad \beta_p \cap \alpha_q \neq \emptyset \Rightarrow \beta_p \subset \alpha_q$

La recherche d'une partition de ce type sera donc entreprise à partir de la partition précédente B'.

On note C' = {ensemble des applications de B' dans N}
Soit τ' une application de l'ensemble C' qui vérifie les propriétés suivantes:

7/ τ' est une surjection de l'ensemble B' dans l'ensemble $\{1, 2 \ldots t\}$

8/ L'image réciproque de $\{1\}$ ne comporte qu'un seul élément noté $\alpha_1 = \{\beta_{1,1}\}$ (cet élément étant lui-même un ensemble)

Soit $q \in \{2,3 \ldots t\}$
On note $\alpha_q = \{\beta_{1,q}, \beta_{2,q}, \ldots \beta_{m_q,q}\}$ l'image réciproque de $\{q\}$ par τ'.

L'ensemble des images réciproques des ensembles $\{1\}, \{2\} \ldots \{t\}$ forme une partition de l'ensemble B' que l'on note: $B'' = \{\alpha_q, q \in 1, 2 \ldots t\}$

ou encore $B'' = \{(\beta_{1,1}), (\beta_{1,2}, \beta_{2,2}, \ldots \beta_{m_2,2}), \ldots (\beta_{1,t}, \beta_{2,t}, \ldots \beta_{m_t,t})\}$

On possède donc la propriété suivante:

$$\text{card}(\alpha_1) + \text{card}(\alpha_2) + \ldots \text{card}(\alpha_t) = 1 + m_2 + m_3 + \ldots + m_t = m$$

On note $(B_1^{p,q}, B_2^{p,q}, \ldots B_{n_{p,q}}^{p,q})$ l'ensemble représenté par $\beta_{p,q}$ et $(h_i^{p,q}, l_i^{p,q})$ le couple qui représente l'élément $B_i^{p,q}$.

L'application τ' est définie de manière à respecter les propriétés suivantes:

9/ L'élément $\beta_{1,1}$ de l'ensemble B' est choisi de manière à avoir

$$\max(l_1^{1,1}, l_2^{1,1} \ldots l_{n_{1,1}}^{1,1}) = l_{max}$$

10/ $\forall q \in 2,3 \ldots t \quad \sum_{i=1}^{m_q} \max(l_1^{i,q}, l_2^{i,q}, \ldots l_{n_{i,q}}^{i,q}) \leq l_{max}$

On obtient ainsi, conformément à la propriété 6/, deux partitions B' et B'' de l'ensemble B, ces deux partitions vérifient respectivement les propriétés 3/, 4/, 5/ et les propriétés 9/, 10/.

On note alors ces partitions de la manière suivante:

$$\{[(B_1^{1,1}, B_2^{1,1}, \ldots B_{n_{1,1}}^{1,1}), (B_1^{1,2}, B_2^{1,2}, \ldots B_{n_{1,2}}^{1,2}), (B_1^{2,2}, B_2^{2,2}, \ldots B_{n_{2,2}}^{2,2}) \ldots$$
$$(B_1^{m_2,2}, B_2^{m_2,2}, \ldots B_{n_{m_2,2}}^{m_2,2}), \ldots (B_1^{1,t}, B_2^{1,t}, \ldots B_{n_{1,t}}^{1,t}), \ldots$$
$$(B_1^{2,t}, B_2^{2,t}, \ldots B_{n_{2,t}}^{2,t}) \ldots (B_1^{m_t,t}, B_2^{m_t,t}, \ldots B_{n_{m_t,t}}^{m_t,t})]\}$$

Exemple:

Soit B' la partition de l'exemple précédent

$B = \{B_1, B_2 \ldots B_8\}$

$B' = \{\beta_1, \beta_2, \beta_3, \beta_4\}$

$\tau' : B' \longrightarrow N$

$\beta_1 \longrightarrow 2$

$\beta_2 \longrightarrow 1$

$\beta_3 \longrightarrow 2$

$\beta_4 \longrightarrow 2$

$\tau'(B') = \{1, 2\}$

La partition correspondante de ' est alors la suivante:

$$\{\alpha_1, \alpha_2\} = \{(\beta_2), (\beta_1, \beta_3, \beta_4)\}$$

Ces deux partitions permettent d'obtenir les groupements suivants:

$$\{[(B_2, B_1)], [(B_3), (B_4, B_5, B_8), (B_6, B_7)]\}$$

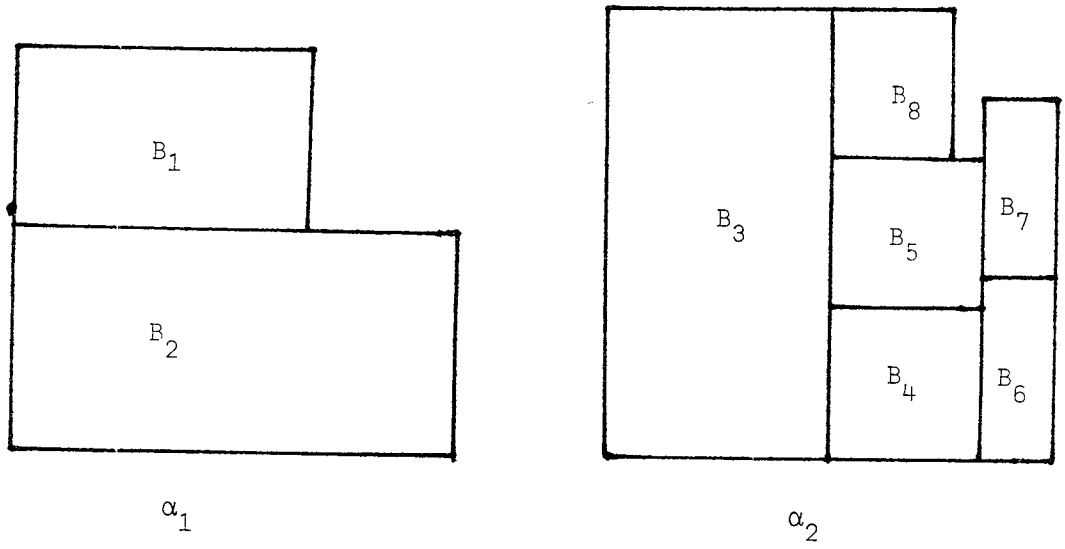


figure 4.4.

4/4.2. Découpage de l'écran

Le découpage de l'écran est effectué suivant trois niveaux. Ces différents niveaux proviennent des groupements imbriqués effectués sur l'ensemble des blocs.

Le calcul des rapports k_1 et k_2 des similitudes à appliquer aux blocs de *B* et à l'écran fictif, est effectué en fonction de la partition *B"*, des valeurs de k_{max} et de l_{max}, et des dimensions de l'écran réel. Soit ε_1 et ε_2 des coéfficients qui permettent d'établir un espacement entre les blocs à placer. On pose $H = h_{max} + \varepsilon_1$ et $L = l_{max} + \varepsilon_2$

4/4.2.1. Découpage de premier niveau

Soit a et b les dimensions d'un rectangle qui représente l'écran fictif. Soit P le plan rapporté à un repère orthonormé, et soit E une partie rectangulaire de P, définie de la manière suivante:

$$E = \{(x,y) \in P : 0 \leq x \leq a \quad \text{et} \quad 0 \leq y \leq b\}$$

Soit (c,d) un couple appartenant à E et (dx, dy) un couple de réels positifs vérifiant la propriété suivante:

$$11/ \quad c+dx \leq a \quad \text{et} \quad d+dy \leq b$$

Soit D un ensemble construit à partir de ces deux couples et défini de la manière suivante:

$$D = \{(x,y) \in P : c \leq x \leq c+dx \quad \text{et} \quad d \leq y \leq d+dy\}$$

On note $\overset{\circ}{D}$ l'ensemble D ouvert, soit:

$$\overset{\circ}{D} = \{(x,y) \in P : c < x < c+dx \quad \text{et} \quad d < y < d+dy\}$$

Cet ensemble D, qui par construction est inclus dans E, est défini par la donnée d'un couple de E et d'un couple de $R^+ \times R^+$, ces deux couples vérifiant la propriété 11/.

On note *D* l'ensemble constitué des ensembles D obtenus en faisant varier ces couples.

Soit f une application injective de *B"* dans *D*

$$f \quad B" = \{\alpha_q, \ q \in 1,2...t\} \longrightarrow D$$

$$\alpha_q \longrightarrow f(\alpha_q) = D_q$$

On note (c_q,d_q) et (dx_q,dy_q) les couples correspondant à la définition de D_q.

L'application f est définie de manière à respecter les propriétés suivantes:

$$12/ \ \forall q \in 1,2...t \ , \ \forall q' \ 1,2...q-1, \ q+1...t \quad \overset{\circ}{D}_q \cap \overset{\circ}{D}_{q'} = \emptyset$$

$$13/ \ \forall q \in 1,2...t \quad dx_q = L \quad \text{et} \quad dy_q = H$$

Exemple:

Soit la partition B'' de l'exemple précédent $B'' = \{\alpha_1, \alpha_2\}$

Les calculs permettent d'obtenir les résultats suivants:

$$L = l_{max} + \varepsilon_1 = 30 + 12 = 42$$

$$H = h_{max} + \varepsilon_2 = 30 + 12 = 42$$

$$a = 84$$

$$b = 42$$

$$f: \{\alpha_1, \alpha_2\} \longrightarrow D$$

$$\alpha_1 \longrightarrow D_1 \text{ avec } \begin{cases} (c_1, d_1) = (0,0) \\ (dx_1, dy_1) = (42,42) \end{cases}$$

$$\alpha_2 \longrightarrow D_2 \text{ avec } \begin{cases} (c_2, d_2) = (42,0) \\ (dx_2, dy_2) = (42,42) \end{cases}$$

$$E = D_1 \cup D_2$$

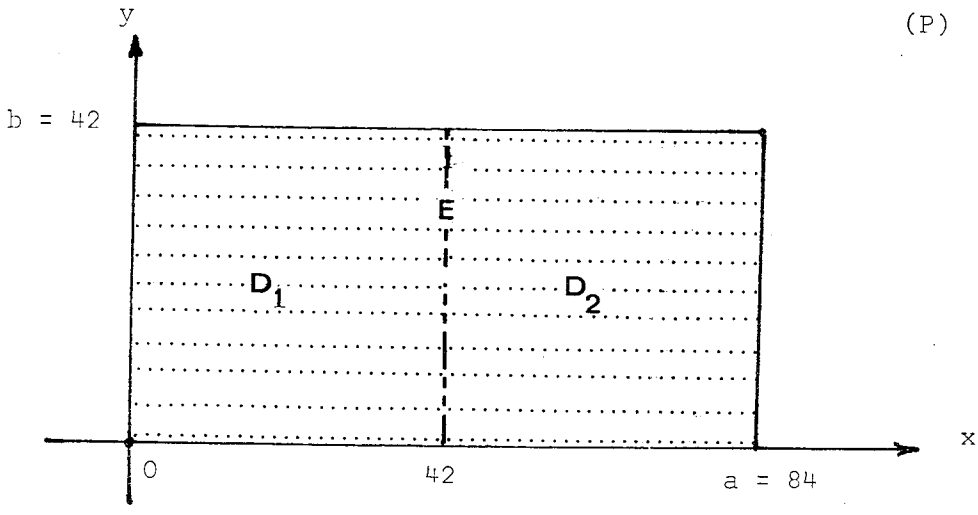


figure 4.5.: représentation du premier découpage

4/4.2.2. Découpage de deuxième niveau

Chaque portion obtenue au découpage précédent est l'objet d'un deuxième découpage.

Soit fq une application injective de α_q dans D

$$f_q : \alpha_q = \{\beta_{1,q}, \beta_{2,q} \ldots \beta_{m_q,q}\} \longrightarrow D$$

$$\beta_{p,q} \longrightarrow f_q(\beta_{p,q}) = D_{p,q}$$

On note F l'ensemble $\{f_q, q \in 1,2 \ldots t\}$

On note $(c_{p,q}, d_{p,q})$ et $(dx_{p,q}, dy_{p,q})$ les couples correspondant à la définition de $D_{p,q}$.

L'application f_q est définie de manière à respecter les propriétés suivantes

14/ $\forall p \in 1,2 \ldots m_q \qquad D_{p,q} \subset D_q$

15/ $\forall p \in 1,2 \ldots m_q, \forall p' \in 1,2 \ldots p-1,p+1 \ldots m_q \quad \overset{\circ}{D}_{p,q} \cap \overset{\circ}{D}_{p',q} = \emptyset$

16/ $\sum_{k=1}^{m_q} dx_{k,q} = dx_q$

17/ $\forall p \in 1,2 \ldots m_q \quad dy_{p,q} = dy_q$

Exemple:

Soit les partitions B'' et B' des exemples précédents

$B'' = \{\alpha_1, \alpha_2\}$

$B' = \{\beta_2, \beta_2, \beta_3, \beta_4\}$

$\alpha_1 = \{\beta_2\}$ et $\alpha_2 = \{\beta_1, \beta_3, \beta_4\}$

$f_1 : \alpha_1 = \{\beta_2\} \longrightarrow D$

$\beta_2 \longrightarrow D_{1,1}$

L'application f_1 ne permet d'effectuer dans ce cas aucun découpage de deuxième niveau (l'ensemble α_1 ne comporte qu'un seul élément β_2).

$$f_2 : \alpha_2 = \{\beta_2, \beta_3, \beta_4\} \longrightarrow D$$

$$\beta_1 \text{ (noté } \beta_{1,2}) \longrightarrow D_{1,2} \quad \text{avec} \begin{cases} (c_{1,2}, d_{1,2}) = (42,0) \\ (dx_{1,2}, dy_{1,2}) = (19,42) \end{cases}$$

$$\beta_3 \text{ (noté } \beta_{2,2}) \longrightarrow D_{2,2} \quad \text{avec} \begin{cases} (c_{2,2}, d_{2,2}) = (61,0) \\ (dx_{2,2}, dy_{2,2}) = (14,42) \end{cases}$$

$$\beta_4 \text{ (noté } \beta_{3,2}) \longrightarrow D_{3,2} \quad \text{avec} \begin{cases} (c_{3,2}, d_{3,2}) = (75,0) \\ (dx_{3,2}, dy_{3,2}) = (9,42) \end{cases}$$

$$D_1 = D_{1,1}$$

$$D_2 = D_{1,2} \cup D_{2,2} \cup D_{3,2}$$

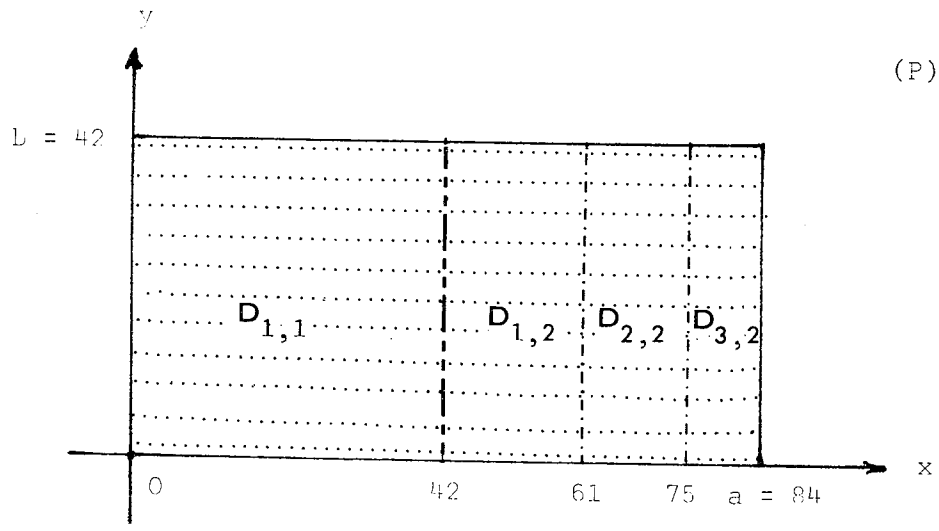


figure 4.6.: représentation du deuxième découpage

4/4.2.3. Découpage de troisième niveau

Chaque portion obtenue au découpage précédent est l'objet d'un troisième découpage.

Soit $f_{p,q}$ une application injective de $\beta_{p,q}$ dans D

$$f_{p,q} : \beta_{p,q} = \{ B_1^{p,q}, B_2^{p,q} \dots B_{n_{p,q}}^{p,q} \} \longrightarrow D$$

$$B_i^{p,q} \longrightarrow f_{p,q}(B_i^{p,q}) = D_{i,p,q}$$

On note F' l'ensemble $\{f_{p,q},\ q \in 1,2\dots t,\ p \in 1,2\dots m_q\}$

On note $(c_{i,p,q}, d_{i,p,q})$ et $(dx_{i,p,q}, dy_{i,p,q})$ les couples correspondant à la définition de $D_{i,p,q}$

L'application $f_{p,q}$ est définie de manière à respecter les propriétés suivantes:

18/ $\forall i \in 1,2\dots n_{p,q} \qquad D_{i,p,q} \subset D_{p,q}$

19/ $\forall i \in 1,2\dots n_{p,q},\ \forall i' \in 1,2\dots i-1,i+1\dots n_{p,q} \qquad \overset{\circ}{D}_{i,p,q} \cap \overset{\circ}{D}_{i',p,q} = \emptyset$

20/ $\forall i \in 1,2\dots n_{p,q} \qquad dx_{i,p,q} = dx_{p,q}$

21/ $\sum_{k=1}^{n_{p,q}} dy_{k,p,q} = dy_{p,q}$

Exemple:

On considère le découpage de l'exemple précédent

$$B'' = \{\alpha_1, \alpha_2\}$$

$$B' = \{\beta_1, \beta_2, \beta_3, \beta_4\}$$

$$B = \{B_1, B_2, \dots B_8\}$$

$\alpha_1 = \{\beta_2\}$ $\qquad$ $\alpha_2 = \{\beta_1, \beta_3, \beta_4\}$

$\beta_1 = \{B_3\}$ $\qquad$ $\beta_2 = \{B_2, B_1\}$

$\beta_3 = \{B_4, B_5, B_8\}$ $\qquad$ $\beta_4 = \{B_6, B_7\}$

- Découpage de $D_{1,1}$:

$f_{1,1}$: $\beta_2 = \{B_2, B_1\} \longrightarrow D$

B_2(noté $B_1^{1,1}$) $\longrightarrow D_{1,1,1}$ avec $\begin{cases}(c_{1,1,1}, d_{1,1,1}) = (0,0)\\ (dx_{1,1,1}, dy_{1,1,1}) = (42, 22{\cdot}5)\end{cases}$

B_1(noté $B_2^{1,1}$) $\longrightarrow D_{2,1,1}$ avec $\begin{cases}(c_{2,1,1}, d_{2,1,1}) = (0, 22{\cdot}5)\\ (dx_{2,1,1}, dy_{2,1,1}) = (42, 19{\cdot}5)\end{cases}$

- Découpage de $D_{1,2}$:

$f_{1,2}$: $\beta_1 = \{B_3\} \longrightarrow D$

B_3(noté $B_1^{1,2}$) $\longrightarrow D_{1,1,2}$ avec $\begin{cases}(c_{1,1,2}, d_{1,1,2}) = (42,0)\\ (dx_{1,1,2}, dy_{1,1,2}) = (19,42)\end{cases}$

- Découpage de $D_{2,2}$:

$f_{2,2}$: $\beta_3 = \{B_4, B_5, B_8\} \longrightarrow D$

B_4(noté $B_1^{2,2}$) $\longrightarrow D_{1,2,2}$ avec $\begin{cases}(c_{1,2,2}, d_{1,2,2}) = (61,0)\\ (dx_{1,2,2}, dy_{1,2,2}) = (14,14)\end{cases}$

B_5(noté $B_2^{2,2}$) $\longrightarrow D_{2,2,2}$ avec $\begin{cases}(c_{2,2,2}, d_{2,2,2}) = (61,14)\\ (dx_{2,2,2}, dy_{2,2,2}) = (14,14)\end{cases}$

B_8(noté $B_3^{2,2}$) $\longrightarrow D_{3,2,2}$ avec $\begin{cases}(c_{3,2,2}, d_{3,2,2}) = (61,28)\\ (dx_{3,2,2}, dy_{3,2,2}) = (14,14)\end{cases}$

- Découpage de $D_{3,2}$:

$f_{3,2}$: $\beta_4 = \{B_6, B_7\} \longrightarrow D$

B_6 (noté $B_1^{3,2}$) $\longrightarrow D_{1,3,2}$ avec $\begin{cases}(c_{1,3,2}, d_{1,3,2}) = (75,0)\\(dx_{1,3,2}, dy_{1,3,2}) = (9,21)\end{cases}$

B_7 (noté $B_2^{3,2}$) $\longrightarrow D_{2,3,2}$ avec $\begin{cases}(c_{2,3,2}, d_{2,3,2}) = (75,21)\\(dx_{2,3,2}, dy_{2,3,2}) = (9,21)\end{cases}$

$D_{1,1} = D_{1,1,1} \cup D_{2,1,1}$

$D_{1,2} = D_{1,1,2}$

$D_{2,2} = D_{1,2,2} \cup D_{2,2,2} \cup D_{3,2,2}$

$D_{3,2} = D_{1,3,2} \cup D_{2,3,2}$

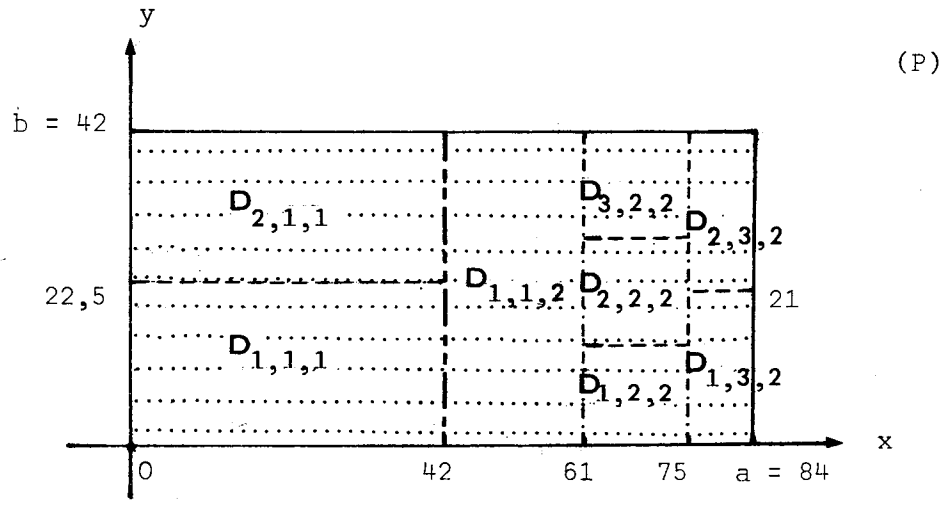


figure 4.7. Représentation du découpage final

La dernière opération consiste alors à appliquer les affinités de rapports k_1 et k_2 et à placer les différents blocs dans leurs zones respectives.

exemple:

Soit un écran rectangulaire de dimensions 168x105 (même unité de mesure que pour les blocs et les zones de découpage).

Les rapports k_1 et k_2 sont alors de 2 et 2,5.

On obtient finalement la répartition suivante:

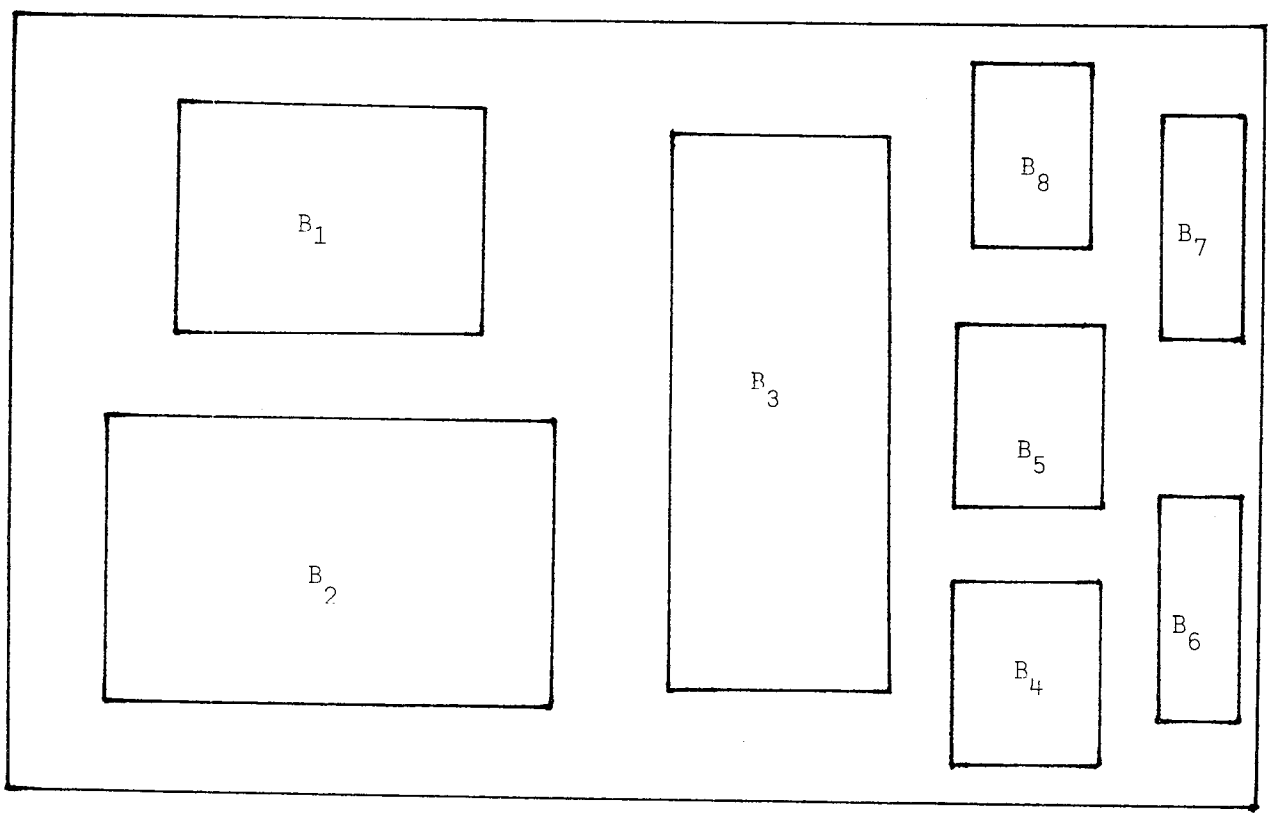


figure 4.8.: Répartition finale

Chapitre 5

LES PRINCIPAUX DOMAINES D'APPLICATION

EXEMPLES D'APPLICATIONS

5.1. LA PEDAGOGIE

5/1.1. Quelques besoins pédagogiques en informatique

La présentation d'un nouveau concept situé totalement hors de l'expérience journalière des étudiants est difficile. D'autre part, l'habitude d'utilisation de ce concept fait oublier à l'enseignant à quel niveau se situent les difficultés de compréhension. Ainsi un point paraissant évident pour l'un, demandera un effort de compréhension pour l'autre. Ce qui différencie principalement l'Informatique des autres sciences telle que la physique ou la chimie, est l'entière nouveauté des concepts et le manque de "démonstrations". En fait, une démonstration ou un ensemble d'images, peut remplacer avantageusement de longues phrases. Si, par exemple, on désire expliquer ce qu'est la mer à quelqu'un qui ne l'a jamais vue, on se perd dans de longues phrases qui ne donnent en fait qu'une perception superficielle de la chose. La principale caractéristique de l'informatique, qui est sa nature dynamique, est souvent mal ressentie par les étudiants. Ceci provient du fait que les divers concepts leur sont présentés d'une manière statique (tableau, explications) et que les manipulations sont impossibles dans les premiers stades. Pour un étudiant, le dynamisme relatif à son programme est totalement masqué: il porte son programme dans la boîte "entrées" et retire le lendemain ses résulats dans la boîte "sorties".

Les techniques d'enseignement usuelles sont souvent insuffisantes en Informatique [P4]:

- la description est souvent incomplète du fait de la nature dynamique du concept à expliquer et de la nature statique du tableau. D'autre part, la nature stricte des mécanismes demande un niveau de précision difficile à atteindre.

- l'appel aux idées similaires est difficile car les concepts traités sont en général totalement différents de l'expérience des étudiants.

- la démonstration demande l'apport de techniques dynamiques. C'est à ce niveau que se situe l'utilité des diverses techniques de production de films et d'images dynamiques.

- le contact est réalisé à travers les programmes. Il est en réalité insuffisant car il masque l'aspect dynamique pour ne traiter que l'aspect: "programme-résultat".

5/1.2. Apport de l'animation calculée comme aide à l'enseignement en Informatique

5/1.2.1. Les films (vidéo tape, microfilms)

La production de films à l'aide d'un système d'animation calculée possède de nombreux avantages sur la méthode conventionnelle (production manuelle de toutes les images intermédiaires). Un film, par exemple, qui est défini mathématiquement ou algorithmiquement serait difficile à expliquer aux professionnels du dessin animé. D'autre part, un programme peut générer un ensemble de films différents en changeant quelques paramètres, alors que la méthode conventionnelle implique la production de films similaires toujours avec le même travail.

Cette méthode permet donc de réduire le coût d'un ensemble de films et de permettre à chacun de produire ses propres films.

5/1.2.2. L'animation calculée interactive (consoles de visualisation)

Cette catégorie de "films" permet d'obtenir des enchaînements dynamiques d'images dépendant directement de la base de données et évoluant en même temps que le programme. Ceci permet d'une part d'expérimenter ses propres données et d'autre part de contrôler par des moyens de dialogue, le déroulement du film. Cependant, nous pouvons faire les remarques suivantes:

- la quantité d'informations que l'on peut mettre sur l'écran est limitée (taille de l'écran, clignotement dans le cas de consoles avec dispositif de régénérescence de l'image).

- l'enchaînement des images est subordonné au temps nécessaire, aux différents calculs et transferts et au matériel employé (un tube mémoire par exemple oblige à reconstruire l'image à chaque modification du dessin). Il est donc important dans ce cas là d'avoir non pas des images "correctes" (détails, ombres, couleurs...), mais des images "approchées" manipulables plus rapidement.

En fait, le choix entre l'une ou l'autre technique dépend du mode d'utilisation du film. Nous citerons par analogie le problème suivant: produire un film afin de montrer les évolutions d'une amibe. Deux solutions sont alors possibles:

- soit produire un film "enregistré" avec du bon matériel et une bonne préparation. On obtient alors un film de bonne qualité: images complètes, nettes, claires, possibilité de cadrage et de zoom...
- soit placer l'amibe sous un microscope afin de projeter la scène sur un écran. Il est évident que dans ce cas là nous obtiendrons une image de moins bonne qualité. Cependant, cette solution permet d'observer une amibe réelle en mouvement. Il est possible en particulier de stimuler le micro-organisme avec des substances chimiques et d'observer les effets à l'instant où ils apparaissent.

5/1.2.3. Aide à l'enseignement

De nombreux domaines de l'informatique sont abordés par la description d'algorithmes plus ou moins complexes (tri, analyse syntaxique, manipulation de tables, gestion mémoire...). Dans la plupart des cas la difficulté ne réside pas dans la complexité de l'algorithme mais dans la dimension des domaines d'application. En effet, les avantages d'une méthode n'apparaissent que lorsque celle-ci est appliquée à un problème de dimension réelle (gestion d'une table par hash-coding par exemple). D'autre part, plusieurs algorithmes peuvent être utilisés pour effectuer une tâche, le choix de la méthode étant fonction de plusieurs paramètres tel que le temps de calcul, la place mémoire utilisée, la complexité de codage ou encore la base de donnée à traiter [P5]. Il est nécessaire dans ce cas de faire une

étude comparative des divers algorithmes afin d'en montrer les avantages et les inconvénients avec les différentes configurations [P6]. En fait, l'efficacité d'un algorithme particulier appliqué sur une base de donnée particulière n'est démontrée que par extrapolation des résultats obtenus sur un exemple de "petite taille". L'utilisation de l'animation calculée doit permettre d'une part de mettre en évidence le dynamisme et de faciliter ainsi la tâche de l'enseignant, et d'autre part de trouver des symbolismes capables de montrer les algorithmes travaillant sur des problèmes de taille réelle [P1].

Le traitement des algorithmes n'est cependant pas le seul aspect abordable par l'animation calculée. La sémantique des programmes (diagrammes, instantanées des états des variables...) [P3], le fonctionnement d'un ordinateur ou d'une machine langage [P4] par exemple peuvent être traités de la même manière.

5/1.2.4. Présentation de concepts à l'aide d'images dynamiques

L'utilisation d'images dynamiques devient un nouvel outil pour l'enseignement possédant des avantages et des inconvénients. La marche à suivre est alors la suivante [P4]:

- identifier et délimiter le nouveau concept,
- décider des exemples les mieux adaptés pour exposer le concept,
- définir les points qui doivent être mis en évidence et à quel niveau de détail.

Ceci amène parfois à "repenser" la procédure d'enseignement pour utiliser au mieux les outils dynamiques (voir "Rethinking procedure" par ELLIS [P7]). Il est par exemple important de montrer le changement dans le contexte où il se place (action primitive et répercussion sur le reste de l'univers). D'autre part, la compréhension d'un processus complexe demande un découpage en plusieurs niveaux:

- étude de l'action spécificque de chaque constituant (sous processus),
- étude des interactions entre les divers constituants.

En fait, l'ordre d'examen des composants dépend de la méthode employée: on peut étudier les sous-processus en détail puis les interactions, ou bien

partir d'une situation critique de l'ensemble afin d'explorer l'action des divers composants. Ceci implique le découpage d'un concept en sous-concepts aussi bien du point de vue théorique que du point de vue de la représentation. Il est donc nécessaire d'avoir plusieurs niveaux de représentation afin de ne pas détourner l'attention du mécanisme que l'on étudie. Par exemple, on élimine le calcul des expressions (niveau de détail supérieur) lors de l'étude du comportement sémantique d'un programme PL/1 dans une boucle [P4].

D'autre part, il est parfois important de pouvoir attirer l'attention sur un point précis afin de "voir" rapidement qu'un changement d'état a eu lieu, ou qu'une action quelconque va être entreprise. Ceci peut être réalisé par exemple par l'apparition d'un symbole précis (curseur, flèche, tirets, caractère non courant sur l'écran...) ou par l'utilisation d'une possibilité technologique (changement de couleur ou de luminosité, clignotement...).

5/1.2.5. Critères intervenant dans le choix du mode de représentation

- Influence du mode d'utilisation du concept sur la représentation:

La représentation choisie doit suggérer à l'interlocuteur le "fonctionnement" du concept présenté. Ainsi, par exemple, une pile d'éléments sera représentée par un contour fermé ne possédant qu'un seul accés (entrées-sorties d'éléments) et une file d'attente par un contour fermé possédant deux accés (une entrée et une sortie pour les éléments). HOPGOOD pour sa part utilise un contour fermé afin de représenter la nature circulaire (accés par clés modulo m) des tables de hash-code [P1].

- Influence du type de concept sur la représentation:

La représentation choisie doit mettre en évidence les renseignements liés au concept présenté. Dans le cas des tables de hash-code [P1], la longueur de la recherche liée à chaque entrée traitée est représentée par une étoile (une seule entrée examinée) ou par n batonnets (n entrées examinées) dans la cellule correspondante à cette entrée. Ceci permet de "voir" rapidement le nombre de recherches minimales (ce qui n'aurait pas été le cas

avec une représentation chiffrée). Pour ce qui est de la sémantique des programmes ([P3], [P4]), nous noterons la représentation classique des variables (nom, type et valeur) et l'utilisation de symboles spéciaux (flèches, tirets, curseur) pour la désignation d'éléments. Lors de la démonstration du mécanisme d'une interruption [P4], le détail des instructions exécutées importe peu et disparait derrière le mécanisme de déroutement (arrivée de l'interruption, échange de PSW...). Par contre, dans le cas de la démonstration du fonctionnement d'une machine de base [P4], il est essentiel de faire apparaître la suite d'instructions et le codage correspondant.

- Influence de la structure liée au concept sur la représentation

La représentation choisie doit montrer la structure liée au concept présenté. Cette structure peut être liée au découpage des données (tableaux, listes...), à l'accés aux variables dans un programme (blocs structurés) ou encore à la nature dynamique du concept (arbre d'appels récursifs d'une procédure par exemple). La sémantique des programmes est souvent abordée par l'étude de diagrammes qui décrivent la structure hiérarchique fréquemment rencontrée. La représentation en arbres, avec parfois des arêtes de fermeture, et la représentation en "contour diagram" ([P8], [P9]) sont souvent utilisées. Le deuxième type de représentation qui est adapté aux blocs structurés permet d'une part, de rendre les relations hiérarchiques évidentes, d'autre part, d'inclure de nombreux renseignements sur le concept présenté.

exemple:

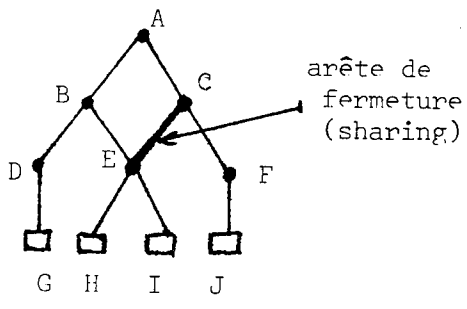


figure 5.1.:représentation en graphe d'une structure hiérarchique

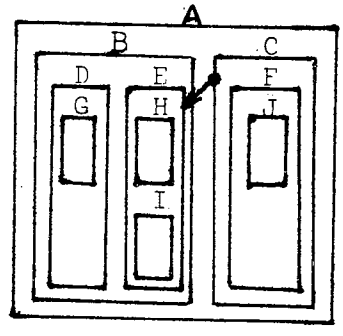


figure 5.2.: représentation en "contour diagram" de la même structure

En réalité, les trois facteurs précédemment présentés (mode d'utilisation, type et structure) interviennent souvent simultanément lors du choix de la représentation. Dans le cas par exemple des tables de hash-code [P1] le mode d'utilisation des tables est suggéré par la représentation circulaire, le type du concept (gestion des tables) est mis en évidence par la représentation des "longueurs de recherche" et la structure interne (table à plusieurs points d'entrées) est indiqué par la concaténation des rectangles représentant les éléments de la table. De même, nous remarquerons dans la présentation de la sémantique des programmes [P3] le découpage en "contour diagram" contenant dans un cas les éléments du programme (structure de base à laquelle on se réfère), dans l'autre, l'état des variables à un instant donné ("instantanée" pris pendant l'exécution).

5.2. AUTRES DOMAINES D'APPLICATION

5/2.1. Diverses classes d'application

Nous ne donnerons évidemment pas de liste complète et définitive des divers domaines susceptibles d'être abordés, mais un ensemble d'exemples d'applications réalisées ou réalisables.

La première catégorie d'applications qui est proche de celle traitée précédemment, concerne l'analyse numérique. A l'heure actuelle, un certain nombre de travaux ont déjà été effectués afin d'utiliser une console de visualisation comme aide à la résolution de problèmes numériques par la méthode des éléments finis. D'autres travaux sont en cours afin d'illustrer des problèmes de convergence (vérification ou contrôle). Cependant, ces divers travaux, bien que se rapportant à des sujets proches, donnent naissance à des réalisations totalement déconnectées et spécifiques. Notre approche permet alors d'illustrer simplement ces algorithmes numériques, le seul travail consistant à définir une représentation pour un ensemble de problèmes (convergence de matrices dans des algorithmes de diagonalisation, de triangularisation,...).

La catégorie d'applications suivante, non moins importante, concerne la simulation de phénomènes physiques. L'intérêt résidant encore dans la séparation de la simulation proprement dite (à l'aide d'un langage de simulation tel que SIMULA 67 par exemple), et de l'interprétation graphique des résulats. Notons d'autre part, que dans ce genre de problème, le nombre

de paramètres qui indiquent l'état transitoire d'un "modèle" est très important et que ces paramètres possèdent de prime abord un aspect totalement hermétique. En effet, il incombe à l'utilisateur lui-même d'interpréter les suites de nombres obtenues, afin de ramener les résultats à son modèle de simulation. Notre approche permet à nouveau de définir une représentation, soit une interprétation des divers paramètres et ceci une fois pour toutes. Cependant, un problème important intervient au niveau de la visualisation: le temps. Ce problème qui n'est pas très important lorsque le modèle fonctionne par rapport à un temps simulé, devient un obstacle majeur dans le cas contraire. Il revêt alors deux aspects totalement indépendants:

- le temps lié à la configuration du système d'exploitation (temps partagé, temps réel, efficacité hardware...),
- et le temps dépendant essentiellement de l'implémentation.

Il est évident, qu'en ce qui nous concerne, nous ne pouvons agir qu'au niveau de l'implémentation. La principale perte de temps étant due à la gestion automatique de l'écran, nous proposons de supprimer cette dernière dans ce cas là. Ceci implique évidemment une description plus précise de la représentation adoptée (taille et position des composants de l'image). D'autre part, une optimisation de l'implémentation, tant du point de vue de l'écriture (assembleur) que des échanges, serait souhaitable. Notons enfin, l'avantage évident qu'aurait,dans ce cas là, l'emploi du calculateur satellite chargé de la gestion graphique.

Nous terminerons en citant les avantages d'un tel système pour tout ce qui est relatif au contrôle des processus, que ce soit en temps réel ou non. L'idée étant alors de "puiser" des données en divers points afin de les visualiser. A nouveau, il est impossible de fournir une réponse précise en ce qui concerne le temps réel, les facteurs intervenant étant souvent difficilement controlables.

5/2.2. L'emploi d'outils graphiques évolués dans certaines applications

L'emploi de certaines représentations "classiques" peut succiter l'utilisation de primitives graphiques évoluées déjà existantes. Il est donc important de pouvoir incorporer le plus facilement possible ces techniques afin de pouvoir les employer ensuite comme des primitives de description graphique.

Nous nous attarderons un peu plus sur un outil graphique évolué très employé, qui est incorporé au système: la visualisation interactive de graphes.
Le problème important relatif à l'incorporation, provenant essentiellement de l'interaction inhérente à l'édition des graphes. En effet, comme dans le cas de la répartition automatique des éléments sur l'écran, il n'existe pas de solution optimale du fait que toutes les contraintes de représentation ne sont pas exprimables mathématiquement. Il est donc nécessaire d'offrir un outil capable de fournir des solutions, ces dernières étant modelables par le biais d'actions purement graphiques. Ces solutions, afin d'être recevables, doivent en premier lieu se soumettre aux contraintes suivantes:

- éviter que les arêtes ne se coupent (planarité formelle et planarité de la représentation),
- et répartir le dessin sur la portion d'écran attribuée afin d'éviter la surcharge de certaines zônes.

D'autres contraintes de type topologique peuvent ensuite intervenir lors de l'édition du graphe, ce qui oblige à considérer non pas une solution générale, mais des solutions adaptées aux différentes applications. Nous citerons, par exemple, le cas des molécules chimiques [I1], des réseaux routiers, des schémas électriques, des graphes de programmes ou encore des réseaux PERT [I2]. Dans ces derniers cas, comme dans celui de la représentation des listes, nous avons adopté une édition basée sur les propriétés de "hiérarchies" (graphes orientés sans circuits), ces propriétés nous permettant de définir une relation d'ordre "verticale" et "horizontale" sur une hiérarchie homomorphe au graphe traité [I2].

5.3.- EXEMPLES D'APPLICATIONS

5/3.1. Illustration d'un programme récursif: "Les tours de Hanoï"

Nous prendrons par exemple un programme écrit en ALGOL W et nous utiliserons dans un premier temps la représentation standard des piles. Notons que, quelque soit les représentations choisies par la suite, il n'y aura pas lieu de retoucher au programme qui se présente sous la forme suivante:

```
BEGIN
 INTEGER N;  READ(N);
 BEGIN
  INTEGER ARRAY PILE(1::3,1::N);
  INTEGER ARRAY PT(1::3);
  PROCEDURE CHANGE(INTEGER VALUE SOURCE,CIBLE);
  BEGIN
   PT(CIBLE):=PT(CIBLE)+1;
   PILE(CIBLE,PT(CIBLE)):=PILE(SOURCE,PT(SOURCE));
   PT(SOURCE):=PT(SOURCE)-1;
  SORTIE(1,SOURCE,CIBLE);
  END CHANGE;
  PROCEDURE HANOI(INTEGER VALUE DEPART,ARRIVEE,N);
   IF N>0 THEN
   BEGIN
    HANOI(DEPART,6-DEPART-ARRIVEE,N-1);
    CHANGE(DEPART,ARRIVEE);
    HANOI(6-DEPART-ARRIVEE,ARRIVEE,N-1);
   END HANOI;
  PROCEDURE SORTIE(INTEGER VALUE NO,SOURCE,CIBLE);
  BEGIN
   STRING(8) ARRAY S(1::3);
   S(SOURCE):="DEPART"; S(CIBLE):="ARRIVEE"; S(6-SOURCE-CIBLE):=" ";
   ECRAN(1,/PILE1/PILE(1,1:PT(1)),S(1),/PILE2/PILE(2,1:PT(2)),S(2),
   /PILE3/PILE(3,1:PT(3)),S(3));
   AFFICHE(NO);
  END SORTIE;
  COMMENT: PROGRAMME PRINCIPAL;
  FOR K:=1 UNTIL N DO PILE(1,K):=K;
  PT(1):=N;  PT(2):=PT(3):=0;
  SEQUENCE;
  SORTIE(1,1,2);
  HANOI(1,2,N);
  SORTIE(100,1,2);
 END;
END.
```

Le descripteur associé étant alors le suivant:

Descripteur 1:

A := Pile(10, cell(5)) ; pointeur A ;

fin.

Nous obtenons alors la représentation de la figure 5.3. Dans le cas où cette représentation n'est pas satisfaisante, nous avons la possibilité d'éditer graphiquement le fichier des descripteurs afin de modifier la description utilisée, et cela pendant l'exécution du programme d'application. Nous pouvons, par exemple, employer la représentation "Tour" définie comme une extension au paragraphe (3.2.6):

```
Descripteur 1:
        A := Tour(10,cellvar(S,10)) ; pointeur A ;
fin.
```

ce qui nous permet d'obtenir la représentation de la figure 5.4.

Notons que toute réédition sera conforme à la dernière représentation choisie, du fait que toute modification de description est enregistrée dans la bibliothèque de descripteurs et que l'emploi d'un descripteur utilisateur de même numéro qu'un descripteur standard annihile l'effet de ce dernier (option classique de l'utilisation des bibliothèques).

Remarque: L'ordre "séquence", qui apparait dans le programme d'application mais qui n'est pas nécessaire dans ce cas là, permet d'indiquer au système qu'il est en présence d'une même image tout au long de l'exécution du programme (pas d'ordre de fin de séquence). Ceci permet au système de ne "calculer" qu'une seule fois l'image et de n'effectuer ensuite que des mises à jour (optimisation de la gestion graphique).

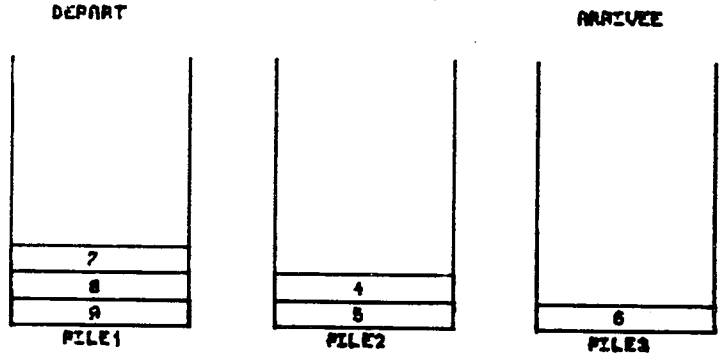


figure 5.3.

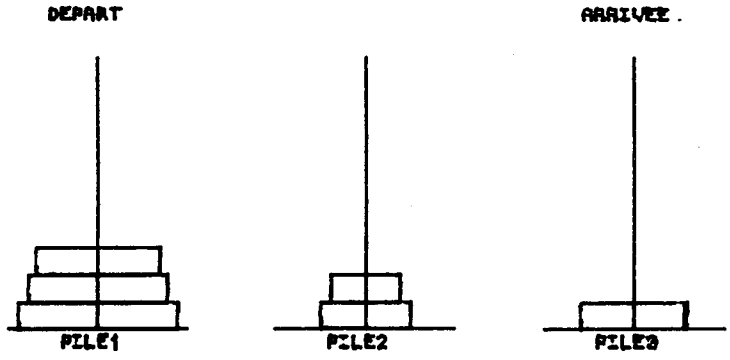


figure 5.4.

Ce type de représentation illustre parfaitement le travail effectué par le programme d'application. Cependant, la forme et le type de l'algorithme employé sont totalement transparents. Il est donc important d'établir un deuxième niveau de visualisation qui reflète la démarche suivie, soit dans notre cas le concept de récursivité. Comme nous l'avons vu précédemment, le programme d'application qui n'a pas accés directement aux piles dynamiques de récursivité, doit simuler lui-même le déroulement récursif (gestion des piles à visualiser et de l'arbre des appels récursifs). Nous rappelons que cette simulation peut être faite automatiquement par un précompilateur évolué.

Le programme d'application prend alors l'aspect suivant:

```
BEGIN
 INTEGER N;  READ(N);
 BEGIN
 INTEGER ARRAY PDEPART,PARRIVEE,PN,PAPPEL(1::30);
 INTEGER ARRAY ARBRE(1::3,1::50);
 INTEGER T1,T2,T3,T4,DERNIER,NOAPPEL;
  INTEGER ARRAY PILE(1::3,1::N);
  INTEGER ARRAY PT(1::3);
  PROCEDURE CHANGE(INTEGER VALUE SOURCE,CIBLE);
  BEGIN
   PT(CIBLE):=PT(CIBLE)+1;
   PILE(CIBLE,PT(CIBLE)):=PILE(SOURCE,PT(SOURCE));
   PT(SOURCE):=PT(SOURCE)-1;
  SORTIE(1,SOURCE,CIBLE);
  END CHANGE;
  PROCEDURE HANOI(INTEGER VALUE DEPART,ARRIVEE,N);
   BEGIN
   T1:=T1+1; PDEPART(T1):=DEPART;
   T2:=T2+1; PARRIVEE(T2):=ARRIVEE;
   T3:=T3+1; PN(T3):=N;
   ARBRE(1,DERNIER):=PAPPEL(T4); ARBRE(2,DERNIER):=DERNIER+1;
    ARBRE(3,DERNIER):=NOAPPEL;
   DERNIER:=DERNIER+1;
   T4:=T4+1; PAPPEL(T4):=DERNIER;
   ECRAN(2,"APPEL",PDEPART(1:T1),PARRIVEE(1:T2),PN(1:T3),PAPPEL(1:T4),
   ARBRE);
   AFFICHE(2);
    IF N>0 THEN
    BEGIN
    NOAPPEL:=2;
     HANOI(DEPART,6-DEPART-ARRIVEE,N-1);
     CHANGE(DEPART,ARRIVEE);
    NOAPPEL:=3;
     HANOI(6-DEPART-ARRIVEE,ARRIVEE,N-1);
    END;
   T1:=T1-1; T2:=T2-1; T3:=T3-1; T4:=T4-1;
   ECRAN(2,"RETOUR",PDEPART(1:T1),PARRIVEE(1:T2),PN(1:T3),PAPPEL(1:T4),
   ARBRE);
   AFFICHE(2);
   END HANOI;
  PROCEDURE SORTIE(INTEGER VALUE NO,SOURCE,CIBLE);
  BEGIN
   STRING(8) ARRAY S(1::3);
   S(SOURCE):="DEPART"; S(CIBLE):="ARRIVEE"; S(6-SOURCE-CIBLE):=" ";
   ECRAN(1,/PILE1/PILE(1,1:PT(1)),S(1),/PILE2/PILE(2,1:PT(2)),S(2),
   /PILE3/PILE(3,1:PT(3)),S(3));
   AFFICHE(NO);
  END SORTIE;
  COMMENT: PROGRAMME PRINCIPAL;
  FOR K:=1 UNTIL N DO PILE(1,K):=K;
 PT(1):=N;  PT(2):=PT(3):=0;
 T1:=T2:=T3:=0;  T4:=1;
 PAPPEL(1):=1; DERNIER:=1;  NOAPPEL:=1;
 FOR K:=1 UNTIL 50 DO ARBRE(1,K):=0;
 SORTIE(1,1,2);
  HANOI(1,2,N);
 SORTIE(100,1,2);
 END;
END.
```

Les descripteurs associés étant alors les suivantes:

```
Descripteur 1:
        A := Tour (10, cellvar (S,10)); pointeur A ;
fin.

Descripteur 2:
        Texte ;
        iter 4: Pile (10, cell(5)) fin ;
        graphe  ⁄ T = 800  * 450 ⁄ ;
fin.
```

Remarques:

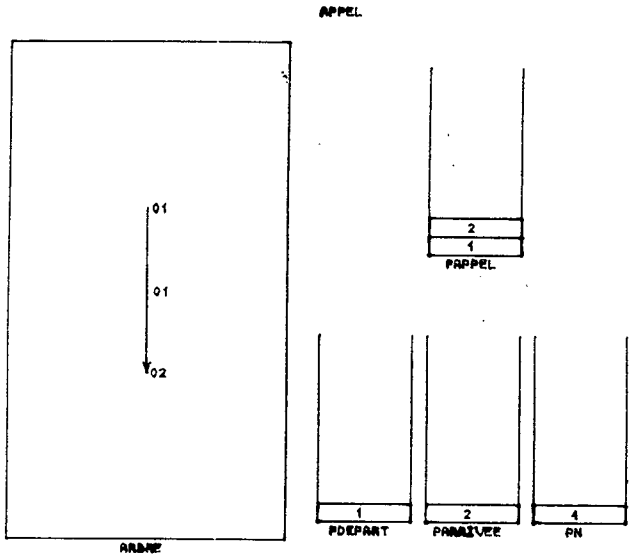
a/ Il aurait été possible d'employer des descripteurs standards unitaires, en "éclatant" l'ordre "écran" correspondant dans le programme d'application.

b/ L'itération permet de ne pas reécrire quatre fois la même unité graphique "pile".

c/ L'indication de taille permet de réserver une large portion d'écran pour la représentation du graphe, et ceci quel que soit l'environnement.

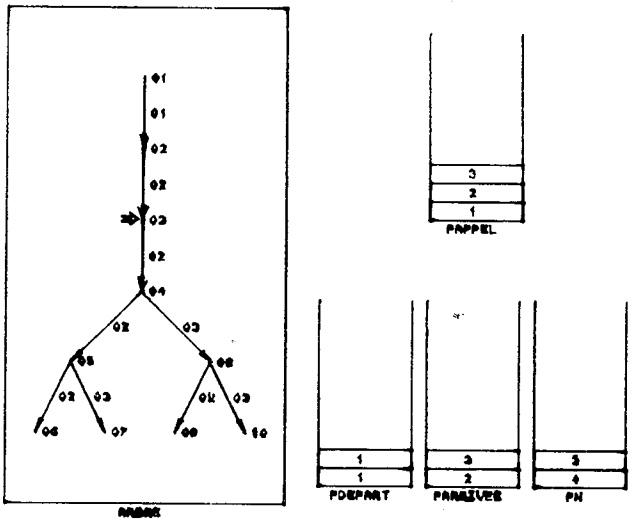
Le premier niveau de représentation correspond à l'illustration précédente alors que le deuxième niveau, que l'on peut visualiser à tout moment par l'intermédiaire des primitives d'interaction, permet d'illustrer la récursivité du programme d'application.

Les dessins qui suivent, représentent quelques images de cette illustration.



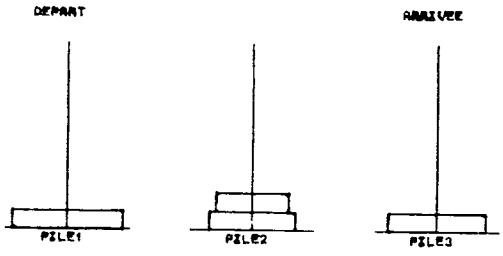
"Pdépart", "Parrivée" et "Pn" représentent les piles des paramètres de la procédure récursive. "Pappel" représente la pile des appels récursifs et "arbre", l'arbre de ces appels. Les sommets de l'arbre représentent les numéros des appels, et les arêtes renseignées l'identification des appels dans le programme.

figure 5.5.
représentation du premier appel



Représentation de l'état de l'algorithme après 10 appels récursifs, dont 7 de terminés.
Le contrôle est alors à "l'intérieur" du corps de procédure relatif au troisième appel.

figure 5.6.
représentation après 10 appels récursifs



Représentation de l'état correspondant des "Tours"
(premier niveau de représentation)

figure 5.7.
représentation de l'état correspondant des "Tours"

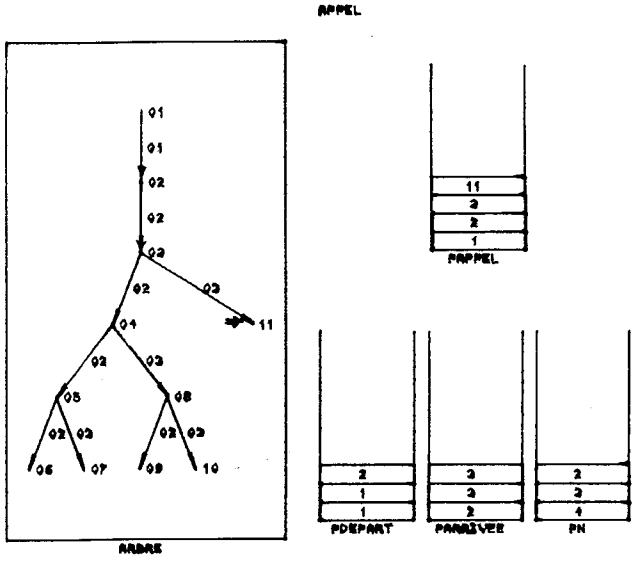


figure 5.8. - 11ème appel récursif

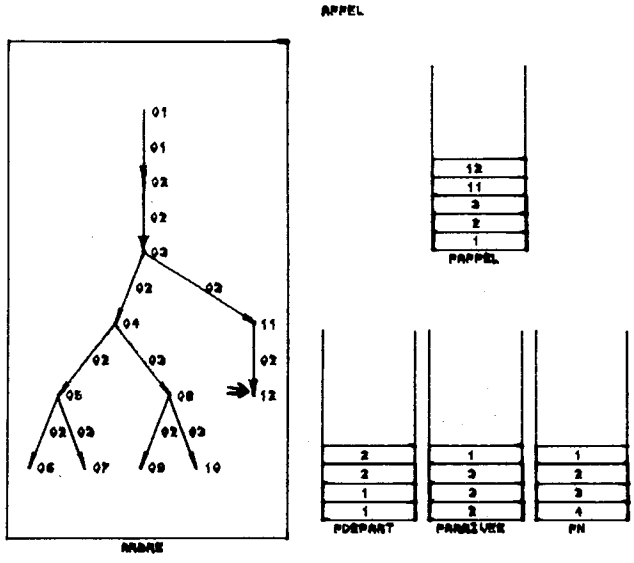


figure 5.9. - 12ème appel récursif

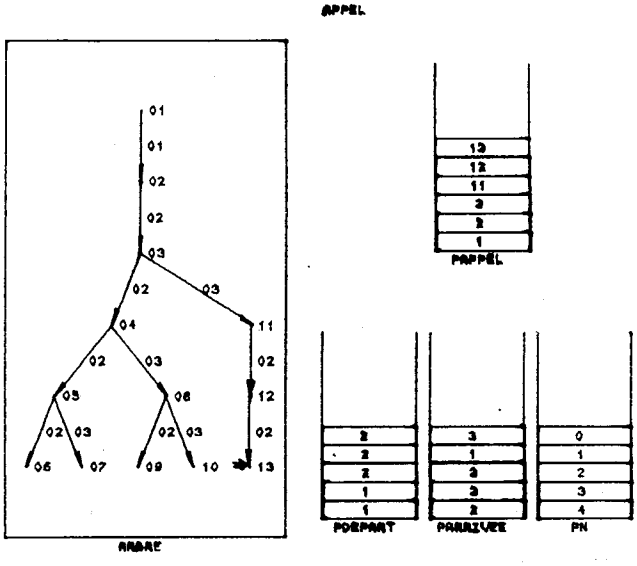


figure 5.10. - 13ème appel récursif

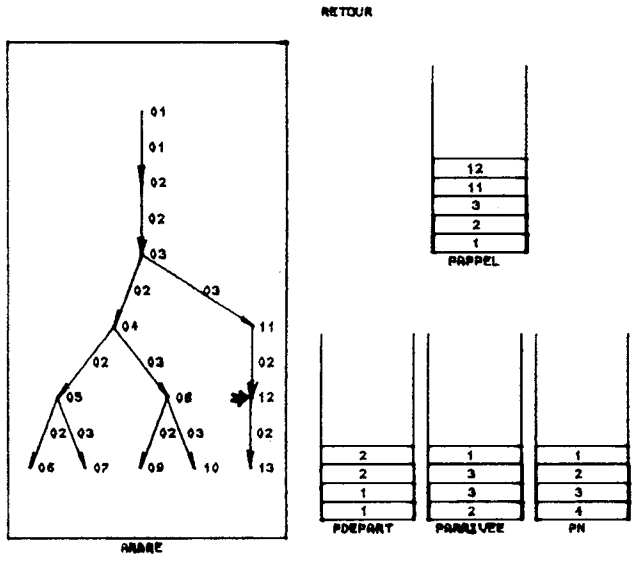


figure 5.11. - retour du 13ème appel récursif

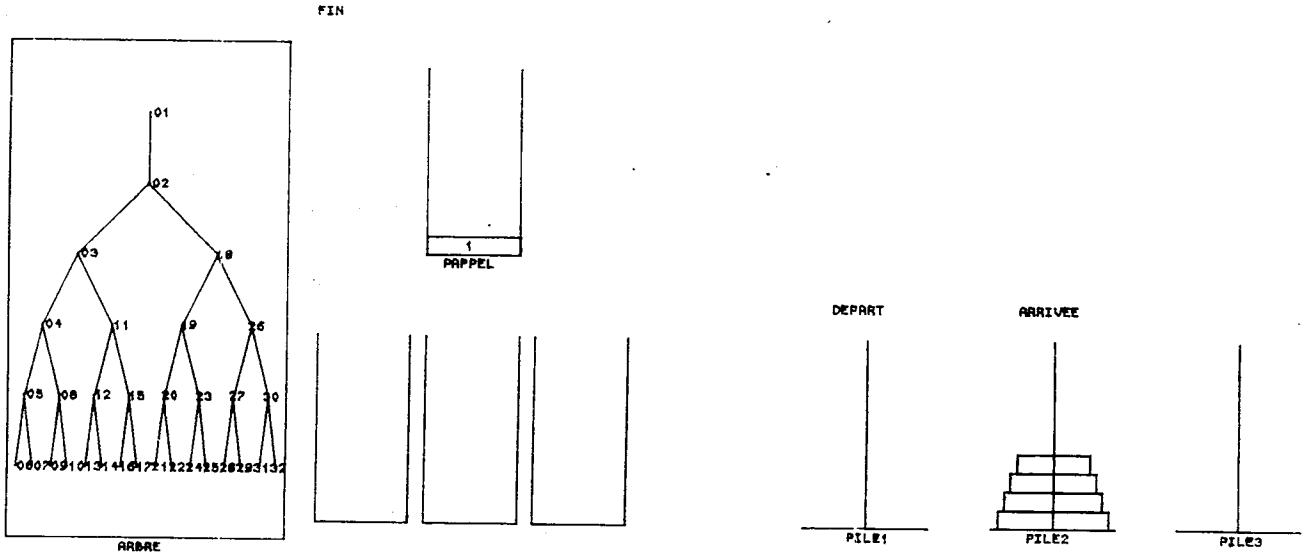


figure 5.12. - images relatives à la fin de l'algorithme

Cet exemple nous a permis entre autre de mettre en évidence l'intérêt existant dans le fait de posséder plusieurs modes de représentations pour les piles. Le premier mode (qui est standard) permet d'avoir une représentation dans laquelle la valeur des données est apparente (valeurs des paramètres empilés, numéros des appels récursifs). Le deuxième type de représentation permet pour sa part de "montrer" rapidement l'emplacement occupé par les éléments (la valeur de ces derniers étant connue).

5/3.2. Simulation du fonctionnement d'un système d'exploitation simple:

Dans cet exemple, nous illustrons le fonctionnement global d'un système d'exploitation simple, ce dernier utilisant le principe des sémaphores (DIJKSTRA). Le programme de simulation permet à l'utilisateur de définir sa propre configuration hardware (nombre d'unités de traitements, nombre de canaux d'entrées-sorties, taille mémoire), cette dernière étant répercutée sur le fonctionnement des trois sémaphores qui gèrent les ressources simulées (mémoire, unités de traitements et canaux d'entrées-sorties).

Les deux opérations "P" et "S" correspondantes, ont alors la forme classique suivante:

s = {Ms, Ps, Cs}

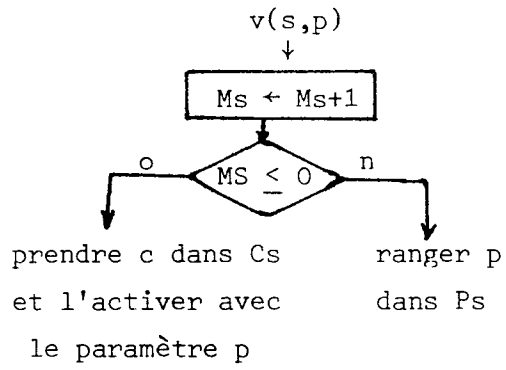


figure 5.13.

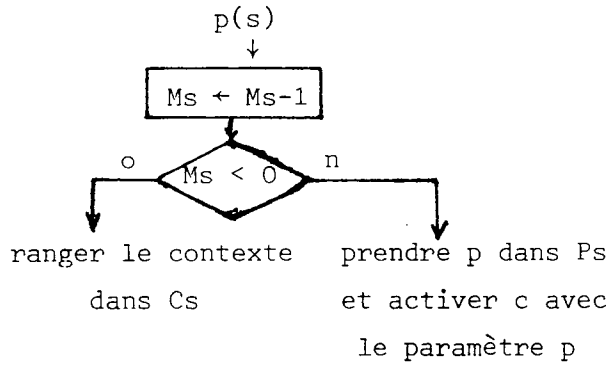


figure 5.14

Avec cette méthode, les variations de la configuration hardware sont répercutées aisément sur les sémaphores, ce qui permet de simuler plusieurs systèmes (monoprocesseur, biprocesseur...).

Partant de là, la visualisation graphique selon notre technique, ne pose ausun problème. En effet, il suffit de visualiser les files d'attente, les ressources et l'écoulement du temps. Cependant afin d'avoir des images plus "parlantes", il est nécessaire d'une part d'employer des représentations

adaptées, et d'autre part d'attirer l'attention de l'utilisateur chaque fois qu'une action est entreprise ou terminée. Nous avons dans ce but employé des flèches qui indiquent à chaque instant les éléments concernés par l'action en cours. Ceci implique évidemment que le programme d'application gère, par un moyen ou un autre, ces différentes flèches. Ce programme, malgré toutes ses possibilités, est relativement court:

```
BEGIN
INTEGER ARRAY ENTREE,IJOB(1::10);
INTEGER ARRAY CPU,CANAL,MEMOIRE(1::4);
INTEGER ARRAY JOB(1::50);INTEGER ARRAY CPUS,C(1::2);
INTEGER S1,S2,S3,NJOB,HORLOGE,NCPU,NC,NMEM,SORTIE,I,J,J2,IJ;
STRING(4) ARRAY FLECHE(1::10);
LOGICAL AFF;
PROCEDURE EDITER(INTEGER VALUE A,B;STRING(74) VALUE C;REAL VALUE D,E);
FORTRAN "EDITER";
INTEGER PROCEDURE NVAL(INTEGER VALUE A,B,C);FORTRAN "NVAL";
PROCEDURE EFFACER(INTEGER VALUE A);FORTRAN "EFFACE";
PROCEDURE GRAPHIC(INTEGER VALUE NO;STRING(4) ARRAY FLECHE(*));
BEGIN
STRING(60) MES;
MES:="0     1     2     3     4     5     6     7     8     9    10";
ECRAN(1,MES,HORLOGE,ENTREE,CPU,CANAL,MEMOIRE,/CPU1/CPUS(1),/CANAL1/C(1),
SORTIE);
FOR K:=1 UNTIL IJ DO ECRAN(1,FLECHE(K));
IF NCPU=2 THEN ECRAN(2,/CPU2/CPUS(2));
IF NC=2 THEN ECRAN(3,/CANAL2/C(2));
AFFICHE(NO);
FOR K:=1 UNTIL 10 DO FLECHE(K):=" ";
END GRAPHIC;
PROCEDURE S(INTEGER ARRAY F(*);INTEGER SS,PP,NP);
BEGIN
COMMENT:FAIT PAR PROCESSEUR (ACTIVE UN PROCESSUS);
SS:=SS-1;IF -SS>NP THEN SS:=-NP;
IF SS>=0 THEN
BEGIN  PP:=F(1); FOR K:=1 UNTIL SS DO F(K):=F(K+1); END;
END S;
PROCEDURE P(INTEGER ARRAY F,PP(*);INTEGER SS,NO);
BEGIN
COMMENT:FAIT PAR PROCESSUS (DEMANDE DE PROCESSEUR);
SS:=SS+1;
IF SS>0 THEN F(SS):=NO ELSE
BEGIN
I:=1;   WHILE PP(I)¬=0 DO I:=I+1;
PP(I):=NO;
END;
END P;
PROCEDURE DISPATCH(INTEGER ARRAY P1,F1,P2,F2(*);INTEGER S1,S2,N1;
INTEGER VALUE N);
BEGIN
FOR K:=1 UNTIL N1 DO
IF P1(K)¬=0 THEN IF JOB(IJOB(P1(K)))=0 THEN
BEGIN
J:=4+K-1+(2*N);
J2:=0;
IJOB(P1(K)):=IJOB(P1(K))+1;
IF JOB(IJOB(P1(K)))¬=-1 THEN P(F2,P2,S2,P1(K)) ELSE
BEGIN
I:=1;   WHILE MEMOIRE(I)¬=P1(K) DO I:=I+1;
MEMOIRE(I):=0;   SORTIE:=P1(K);
J2:=9;
END;
```

```
 P1(K):=C;
 IF J2=0 THEN
 BEGIN  IF S2>0 THEN J2:=3-N ELSE J2:=6+I-1-(2*N);  END;
 FLECHE(J):=FLECHE(J2):="==>";
 S(F1,S1,P1(K),N1);
 IF S1>=0 THEN FLECHE(2+N):="==>";
 GRAPHIC(2,FLECHE);
 END;
 END DISPATCH;
 T:=S1:=NJOB:=J:=MEMOIRE(4):=SORTIE:=C;  I:=-1;
 CPUS(1):=CPUS(2):=C(1):=C(2):=MEMOIRE(1):=MEMOIRE(2):=MEMOIRE(3):=0;
 EDITER(2,1,"/QUELLE CONFIGURATION DESIREZ-VOUS?/",1.,50.);
 EDITER(2,2,"/NOMBRE DE 'CPUS':/,Z2,X10,/NOMBRE DE CANAUX:/,Z2",1,40);
 EDITER(2,3,"/TAILLE MEMOIRE:/,Z2",1,30);
 NCPU:=NVAL(2,2,1);  NC:=NVAL(2,2,2);  NMEM:=NVAL(2,3,1);
 EFFACER(2);
 FOR K:=1 UNTIL 10 DO FLECHE(K):=" ";
 IF (NCPU=2)AND(NC=2) THEN IJ:=8 ELSE IJ:=9;
 S2:=-NCPU;   S3:=-NC;
 WHILE I¬=0 DO
 BEGIN
 IF I=-1 THEN
 BEGIN
 NJOB:=NJOB+1;   IJOB(NJOB):=J+1;  J2:=1;   EFFACER(2);
 EDITER(2,1,"/CONFIGURATION DU JOB:/,Z2,Z2,Z2,Z2,Z2,Z2,Z2,Z2,Z2",1,50);
 END;
 I:=NVAL(2,1,J2);   J:=J+1;   JOB(J):=I;  J2:=J2+1;
 END;
 EFFACER(2);
 NJOB:=NJOB-1;
 FOR K:=1 UNTIL NJOB DO P(ENTREE,MEMOIRE,S1,K);
 FLECHE(1):="==>";
 WHILE SORTIE¬=NJOB DO
 BEGIN
 SORTIE:=0;
 AFF:=FALSE;
 GRAPHIC(1,FLECHE);
 FOR K:=1 UNTIL NMEM DO   IF MEMOIRE(K)=0 THEN
 BEGIN   AFF:=TRUE;
  S(ENTREE,S1,MEMOIRE(K),NMEM);IF S1>=0 THEN FLECHE(1):=FLECHE(8):="==>";
 IF MEMOIRE(K)¬=0 THEN
 BEGIN
 P(CPU,CPUS,S2,MEMOIRE(K));
  IF S2>0 THEN J:=2 ELSE J:=4+I-1;
   FLECHE(8):=FLECHE(J):="==>";
 END;
 END;
 IF AFF THEN GRAPHIC(2,FLECHE);
 DISPATCH(C,CANAL,CPUS,CPU,S3,S2,NC,1);
 DISPATCH(CPUS,CPU,C,CANAL,S2,S3,NCPU,0);
 T:=T+1;     IF T>10 THEN T:=1;
 FOR K:=1 UNTIL NCPU DO
 IF CPUS(K)¬=0 THEN JOB(IJOB(CPUS(K))):=JOB(IJOB(CPUS(K)))-1;
 FOR K:=1 UNTIL NC DO
 IF C(K)¬=0 THEN JOB(IJOB(C(K))):=JOB(IJOB(C(K)))-1;

 END;
 FLECHE(9):="==>";
 GRAPHIC(100,FLECHE);
 END.
```

Une description graphique complète avec taille et positions des représentations, prend par exemple l'aspect suivant:

```
DESCRIPTEUR 1:
TEXTE &T=14*840,P=10*940&;
CELLVAR(S,10)&T=25*840,P=10*900&;
FILE(10,CELL(L,3))&T=650*100,P=120*150&;
FILE(4,CELL(L,3))&T=300*100,P=340*500&;
FILE(4,CELL(L,3))&T=300*100,P=560*500&;
VECTEUR(CELL(M,3))&T=300*200,P=730*100&;
CELL(M,3)&T=60*100,P=340*350&;
CELL(M,3)&T=60*100,P=560*350&;
CELL(M,3)&T=60*100,P=120*100&;
TEXTE &T=14*56,P=90*760&;
TEXTE &T=14*56,P=310*760&;
TEXTE &T=14*56,P=530*760&;
TEXTE &T=14*56,P=310*380&;
TEXTE &T=14*56,P=310*130&;
TEXTE &T=14*56,P=530*380&;
TEXTE &T=14*56,P=530*130&;
TEXTE &T=14*56,P=720*250&;
TEXTE &T=14*56,P=90*130&;
FIN.
DESCRIPTEUR 2:
CELL(M,3)&T=60*100,P=340*100&;
FIN.
DESCRIPTEUR 3:
CELL(M,3)&T=60*100,P=560*100&
FIN.
$
```

Il est évident qu'il est possible d'utiliser une description graphique plus simple, laissant alors au système le soin d'effectuer lui-même la mise en place sur l'écran. On obtient alors, par exemple, la description graphique suivante (qui implique de modifier la forme de passage des données pour les flèches):

```
Descripteur 1:
        A := cellvar (S,10) ; pointeur A ;
   end.
Descripteur 2:
        A := file (10,cell(L,3)) ; pointeur A ;
   end.
Descripteur 3:
        A := vecteur (cell(M,3)) ; pointeur A ;
   end.
```

```
Descripteur 4:
        A := cell (M,3) ; pointeur A ;
    end.
```

Remarque:

La description graphique "pointeur" employée avec une donnée de type chaîne, permet d'attacher un texte à une représentation, et ainsi d'obtenir la juxtaposition sur l'écran du texte et de la représentation impliquée. Cette même description graphique, employée avec une donnée de type entier, permet de "déplacer" une flèche devant la représentation considérée.

Ce programme "illustré" permet donc d'obtenir différentes simulations des fonctions de base d'un système d'exploitation simple. L'utilisateur pouvant dans ce cas "voir" directement les effets provoqués par les modifications qu'il appporte (configuration hardware, type des jobs à traiter,...). D'autre part, l'incorporation de deux niveaux de visualisation, permet d'obtenir soit une image lors de chaque action entreprise ou terminée et ceci indépendamment du temps, soit une image à chaque unité de temps écoulée (regroupant ainsi les diverses actions dans le temps afin de montrer le parallélisme existant).

- exemple d'utilisation

```
quelle configuration désirez-vous?

nombre de CPUs: 1
nombre de CANAUX: 2
taille mémoire: 3
```

figure 5.15 - première image

```
configuration des jobs:

1  3  2  -1
1  2  .....
```

figure 5.16 - deuxième image

Dans les deux images précédentes les réponses sont faites à l'aide du clavier alpha-numérique de la console graphique. Les configurations des jobs sont données par les alternances des temps de calcul et du temps d'entrées-sorties.

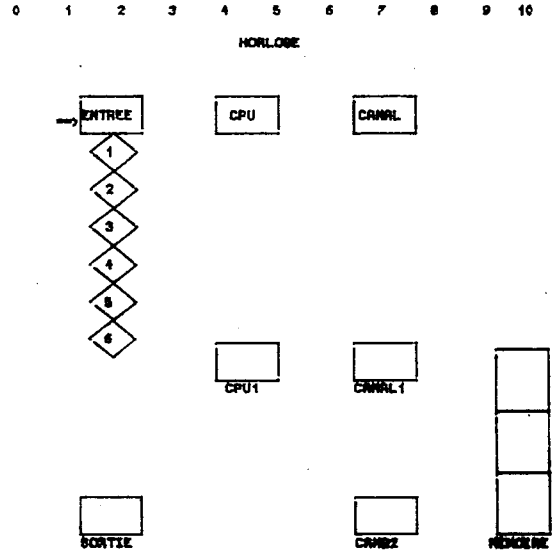


figure 5.17: image initiale

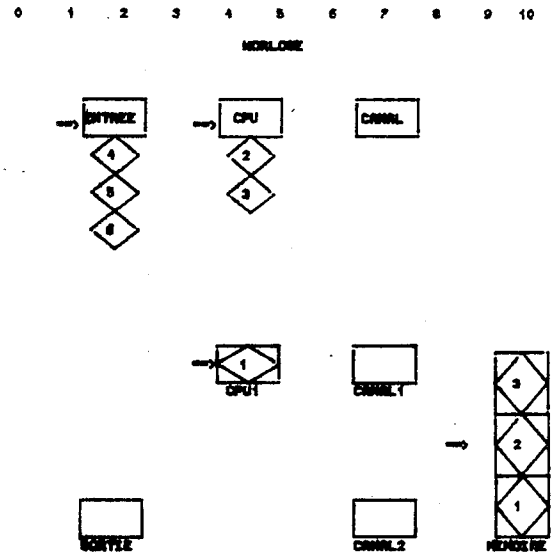
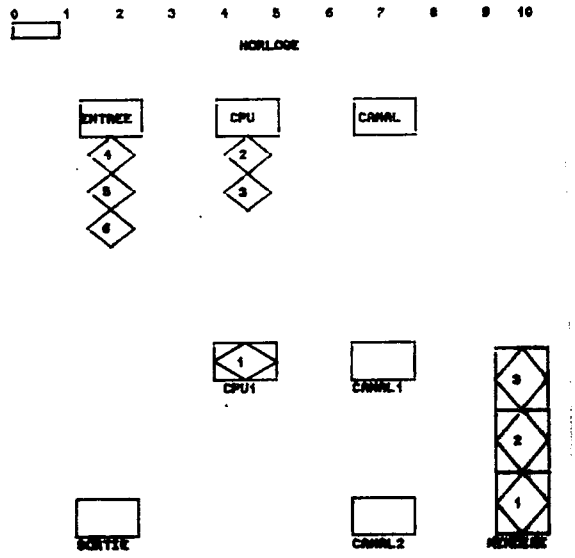


figure 5.18: entrée des trois premiers jobs



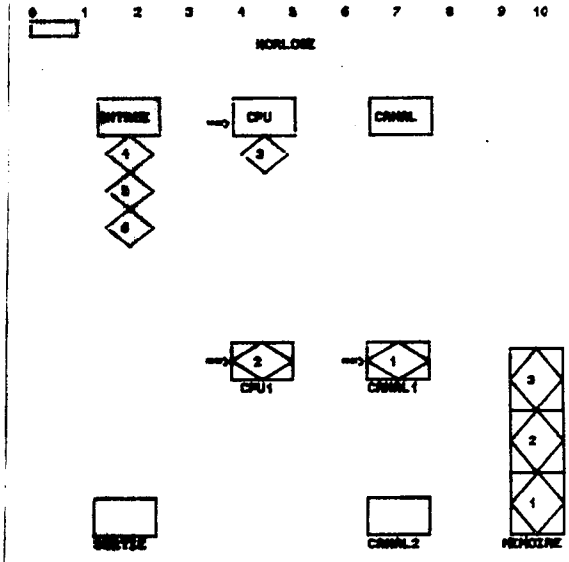
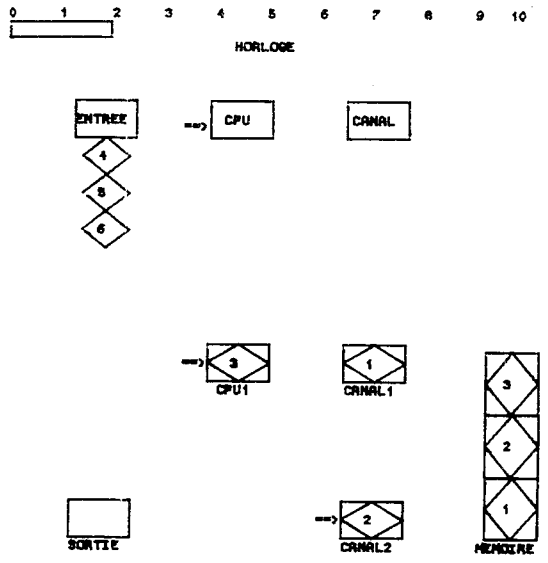


figure 5.19: détail du comportement des jobs après une unité de temps

Le job 1 est pris en charge par le canal numéro un et le job 2, qui était en attente de traitement, est pris en charge par le CPU.



Le job 2 est pris en charge par le canal numéro deux car le canal numéro un est occupé. Le job 3 qui était en attente de traitement est pris en charge par le CPU.

figure 5.20: images après deux unités de temps

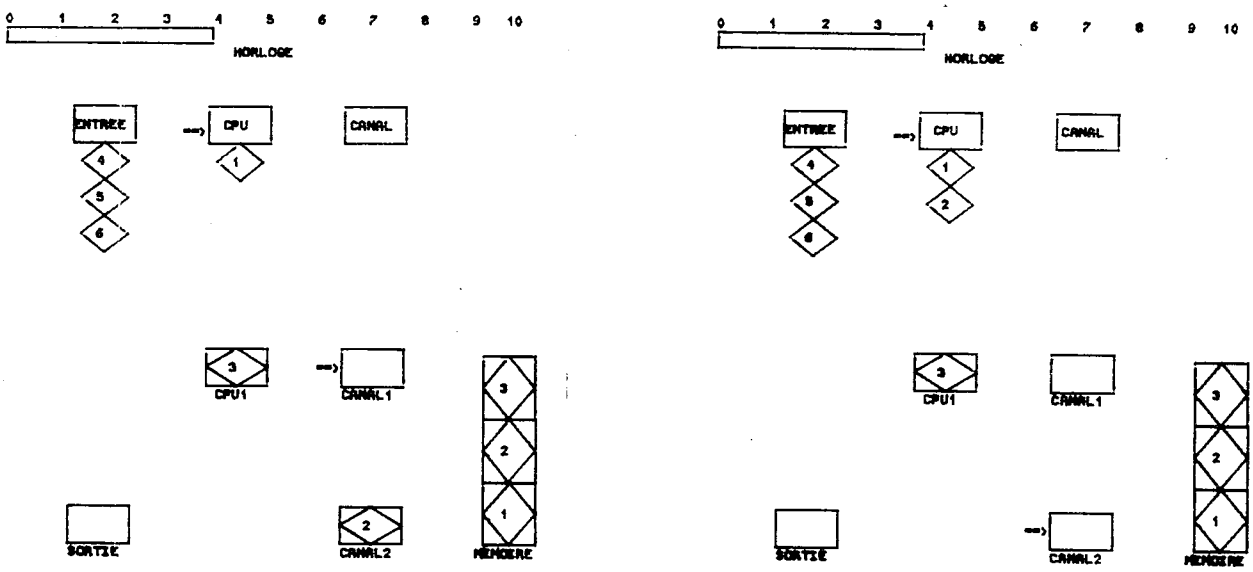


figure 5.21: configuration après quatre unités de temps

Les entrées-sorties des jobs 1 et 2 sont terminées. Ces jobs retournent dans la file d'attente du CPU, ce dernier étant occupé par le job 3.

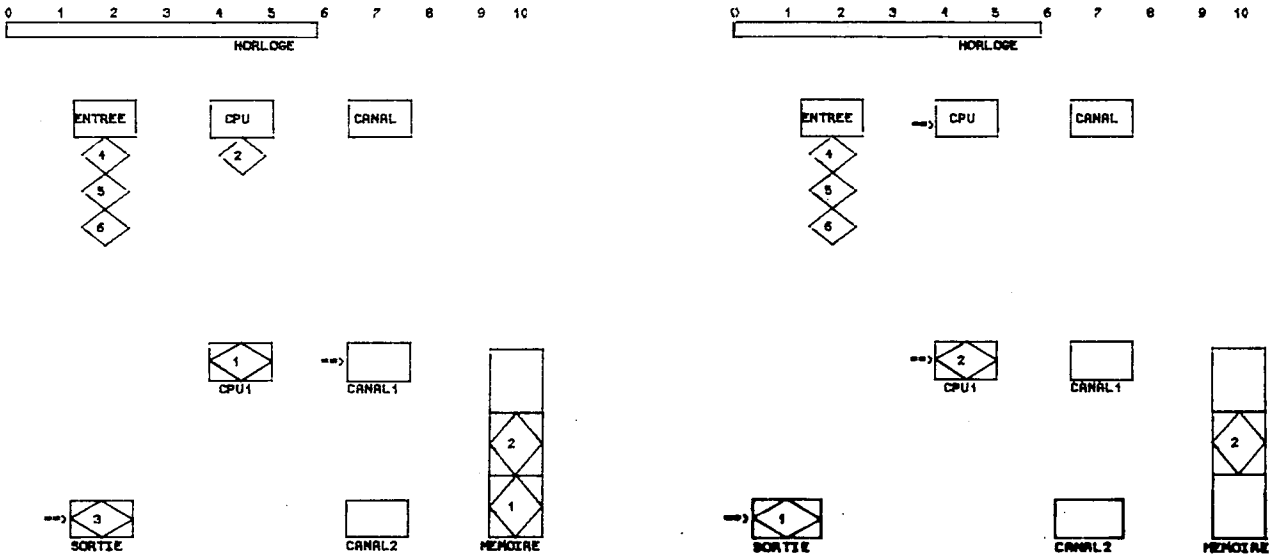


figure 5.22: configuration après six unités de temps

Après six unités de temps, le job 3 se termine (après une entrée-sortie effectuée sur le canal numéro un), ainsi que le job 1 qui était en traitement dans le CPU. Le job 2 est alors pris en charge par le CPU ainsi libéré.

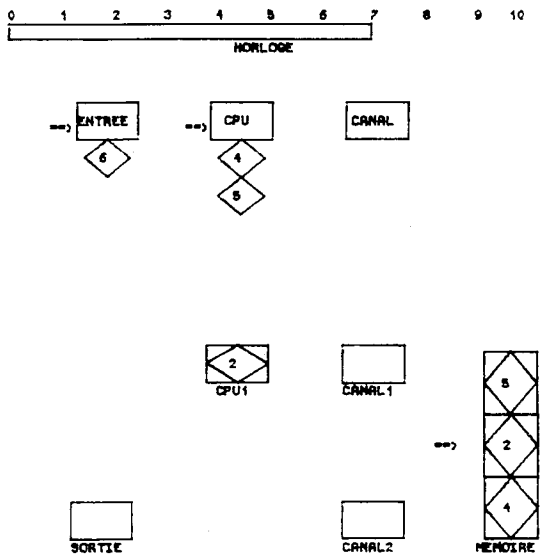


figure 5.23: configuration après sept unités de temps

L'unité de temps suivante, les jobs 4 et 5 ont remplacé en mémoire les jobs 1 et 3...

5/3.3. Illustration d'un algorithme de parcours dans un graphe: "L'algorithme de Yen"

Dans cet exemple, nous allons employer les descripteurs unitaires standards suivants:

descripteur 1: représentation d'un texte,
descripteur 2: représentation d'une variable simple,
descripteur 3: représentation d'un tableau,
descripteur 4: représentation d'un index attaché au dernier tableau généré,
descrtipeur 5: représentation d'un graphe.

Nous employons aussi, dans certains cas, un descripteur de numéro zéro, permettant de rentrer un graphe à l'aide de la tablette sylvania.

L'emploi de ces descripteurs unitaires standards permet de ne faire aucune description graphique, mais oblige à incorporer au programme d'application un plus grand nombre d'ordres "écran" unitaires.

Nous remarquerons également la présence non obligatoire de deux niveaux de visualisation liés aux boucles itératives, et d'une ordre "séquence" qui permet d'indiquer la présence d'une seule image tout au long de l'exécution du programme d'application.

Nous obtenons alors le programme suivant:

```
BEGIN
INTEGER ARRAY M,D(1::20,1::20);
INTEGER I1,I2,X0,L,J,JST,N,I3;
INTEGER ARRAY V,P(1::20);
FOR K:=1 UNTIL 20 DO
FOR KK:=1 UNTIL 20 DO
BEGIN
D(K,KK):=10000000;
M(K,KK):=0;
END;
I1:=0;
WHILE I1¬=50 DO
   BEGIN
   READ(I1,I2,I3);
   IF I1=50 THEN N:=I2 ELSE    D(I1,I2):=M(I1,I2):=I3;
   END;
IF N=0 THEN ECRAN(0,M);
READ(N,X0);
SEQUENCE;
ECRAN(5,/GRAPHE/M);
ECRAN(1,"ALGORITHME DE YEN");
ECRAN(2,X0);
FOR I:=1 UNTIL N DO
   BEGIN
   P(I):=I;
   V(I):=10000000;
   END;
P(X0):=N;  V(X0):=0;  L:=X0;
ECRAN(3,P,V);
AFFICHE(1);
FOR K:=N-1 STEP -1 UNTIL 1 DO
   BEGIN
   JST:=1;
   ECRAN(3,P,V); ECRAN(4,L);
   ECRAN(2,K);
   AFFICHE(1);
   FOR I:=1 UNTIL K DO
      BEGIN
      J:=P(I);
      ECRAN(3,P); ECRAN(4,I,JST);
      ECRAN(3,V); ECRAN(4,L,J);
      ECRAN(2,K);
      AFFICHE(1);
      IF V(J)>V(L)+D(L,J) THEN V(J):=V(L)+D(L,J);
      IF V(P(JST))>V(J) THEN JST:=I;
      END;
   L:=P(JST);
   P(JST):=P(K);
   END;
ECRAN(3,P,V);
AFFICHE(100);
END.
```

La première image que nous obtenons, après avoir lu le graphe à traiter ainsi que le numéro du sommet de départ de l'algorithme, est la suivante:

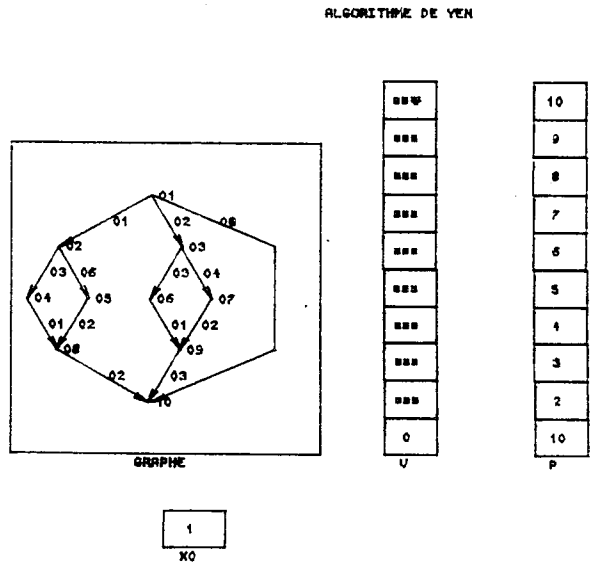


figure 5.24: image initiale

Après avoir utilisé certaines primitives d'interaction, on obtient la représentation suivante (représentation symétrique du graphe):

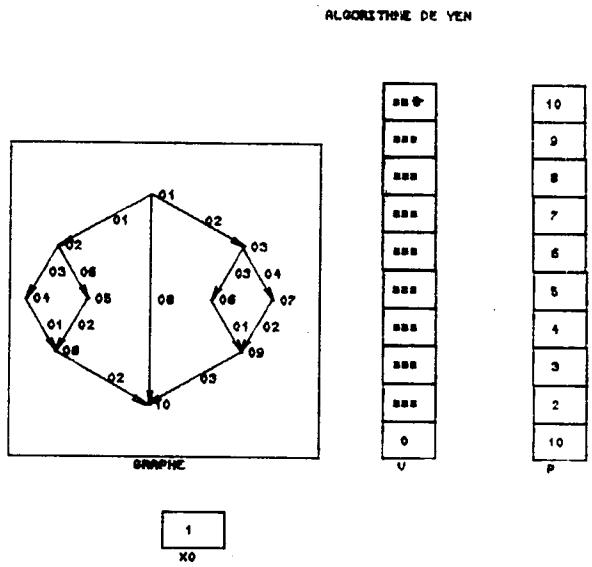


figure 5.25: image initiale modifiée

Les deux images qui suivent, illustrent le premier pas de l'algorithme à deux niveaux différents (boucle englobante et boucle englobée):

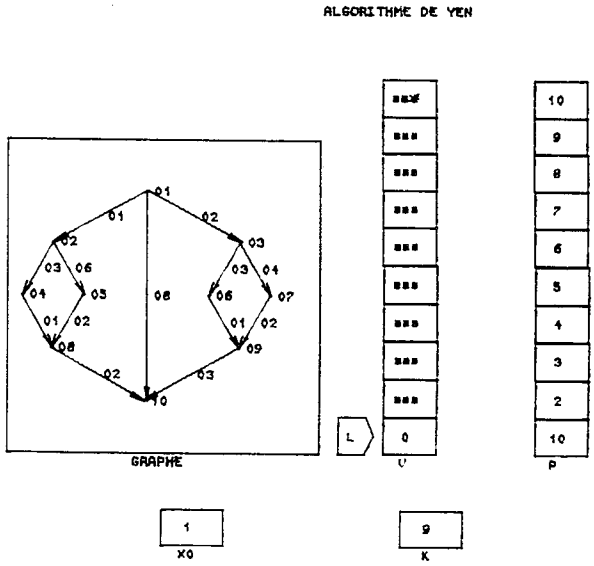


figure 5.26: initialisation de l'algorithme

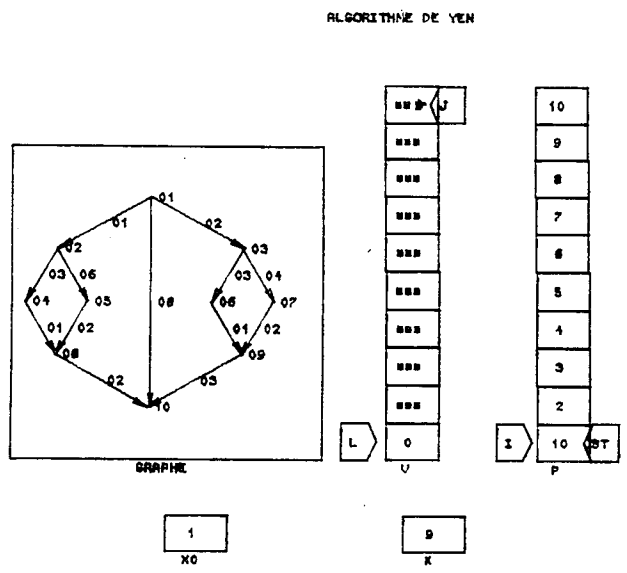


figure 5.27: premier pas de l'algorithme

. . . . .

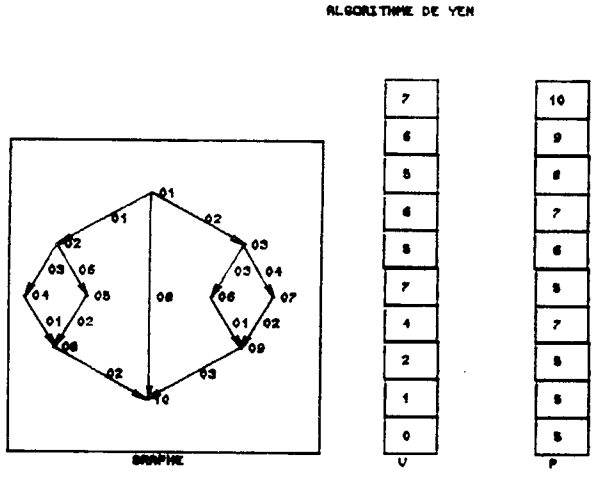


figure 5.28: image finale avec le résultat dans "V"

5/3.4. Illustration d'un algorithme de tridiagonalisation de matrice :

Nous avons pris comme exemple un algorithme de tridiagonalisation de matrices réelles symétriques. Cet algorithme qui fonctionne par itérations successives, permet d'obtenir une matrice tridiagonale qui sera ensuite traitée par un algorithme de diagonalisation (afin d'accélérer la diagonalisation). Le programme d'application correspondant à la tridiagonalisation est alors le suivant:

```
BEGIN
  COMMENT: CE  PROGRAMME REALISE UNE ITERATION DE LA METHODE
     DES ROTATIONS PLANES DANS L'ALGORITHME DE TRIDIAGONALISATION
     D'UNE MATRICE REELLE SYMETRIQUE;
  INTEGER N: READ(N);
  BEGIN
   REAL ARRAY MAT(1::N,1::N);
   REAL C,S,GAM,AUX1,AUX2;  INTEGER IP1;
   FOR I:=1 UNTIL N DO
   FOR J:=1 UNTIL N DO READON(MAT(I,J));
   FOR J:=1 UNTIL N-2 DO
   FOR I:=N-1 STEP -1 UNTIL J+1 DO
   BEGIN
     IF MAT(I+1,J)=0 THEN GOTO FIN;
     IP1:=I+1;
     AUX1:=MAT(I,J);AUX2:=MAT(IP1,J);
     GAM:=SQRT(AUX1*AUX1+AUX2*AUX2);
     IF GAM=0 THEN GOTO FIN;
     IF AUX1<0 THEN GAM:=-GAM;
     MAT(I,J):=MAT(J,I):=GAM;
     MAT(IP1,J):=MAT(J,IP1):=0;
     C:=AUX1/GAM;S:=AUX2/GAM;
     FOR K:=J+1 UNTIL N DO
           BEGIN
           AUX1:=MAT(I,K);AUX2:=MAT(IP1,K);
           MAT(I,K):=C*AUX1+S*AUX2;
           MAT(IP1,K):=S*AUX1-C*AUX2
           END;
     AUX1:=MAT(I,I);AUX2:=MAT(I,IP1);
     MAT(I,I):=C*AUX1+S*AUX2;
     MAT(IP1,IP1):=S*MAT(IP1,I)-C*MAT(IP1,IP1);
     MAT(IP1,I):=MAT(I,IP1):=S*AUX1-C*AUX2;
     FOR K:=J+1 UNTIL N DO
           BEGIN
           MAT(K,I):=MAT(I,K);MAT(K,IP1):=MAT(IP1,K);
           END;
   FIN:ECRAN(1,MAT);
   AFFICHE;
  END;
 END;
END.
```

Si on emploie la représentation standard des matrices, on obtient le type d'image suivant:

2	2	0	1	0	0	0
2	3	3	2	0	0	0
0	3	3	1	2	0	0
1	2	1	3	2	1	0
0	0	2	2	3	0	1
0	0	0	1	0	3	2
0	0	0	0	1	2	2

M A T

figure 5.29: représentation standard de la matrice

Mais cette représentation ne permet pas de suivre facilement les évolutions de la matrice au cours de l'algorithme. Il est en effet plus intéressant d'avoir dans ce cas une représentation moins précise, mais plus "parlante". Nous utiliserons alors une extension du langage de description, permettant de représenter une matrice sous forme d'une nappe comportant des " pics " dont les tailles varient en fonction des valeurs des éléments de cette matrice. Notons qu'il est toujours possible de revenir à la représentation précédente lorsqu'on désire avoir les valeurs exactes de la matrice au cours d'une étape quelconque.

Les images qui suivent illustrent quelques pas de l'algorithme travaillant sur la matrice ci-dessus (les algorithmes concernés ayant été communiqués par M.COSNARD).

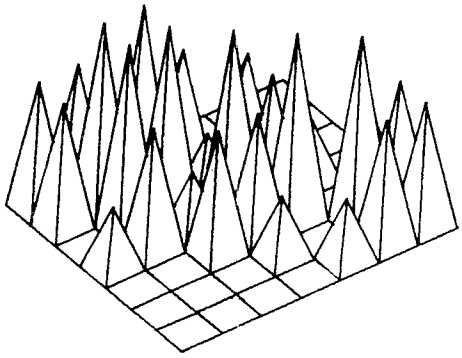


figure 5.30

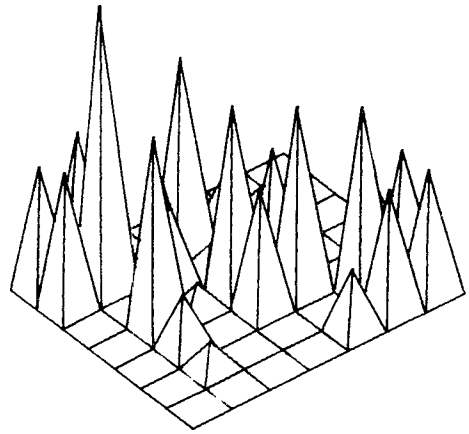


figure 5.31

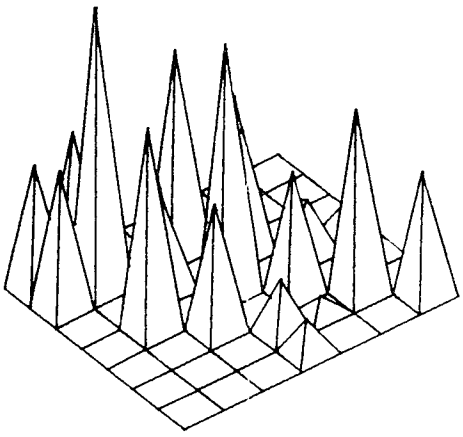


figure 5.32

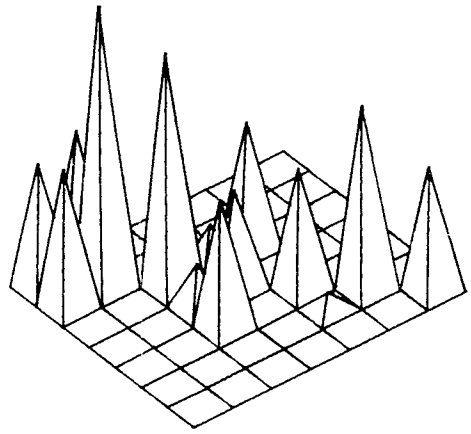


figure 5.33

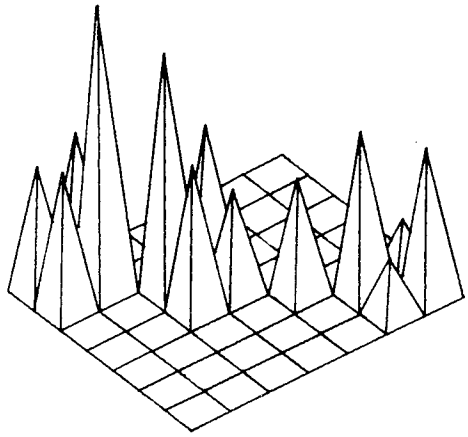


figure 5.34: image finale

CONCLUSION

Cette étude nous a donc permis d'apporter des solutions à un certain nombre de problèmes relatifs à l'utilisation classique des consoles de visualisation, notamment en ce qui concerne la simplicité d'emploi et l'indépendance vis-à-vis de l'application. L'intérêt de l'étude ne réside alors pas dans la forme exacte des solutions adoptées, qui sont parfois discutables, mais dans l'inventaire des mécanismes et des notions nécessaires. Nous avons, en effet, dégagé une structure de base pour ce type de système, ainsi qu'un ensemble de mécanismes généraux à mettre en oeuvre. Afin d'apporter une certaine crédibilité aux diverses assertions, nous avons réalisé d'une manière plus ou moins efficace suivant le cas, l'ensemble des mécanismes introduits, ces réalisations permettant alors de montrer la viabilité de ce type de système.

Par la suite, le succès relatif à divers exemples d'applications, nous a permis de mesurer plus précisément l'intérêt de ce type de système, notamment en ce qui concerne l'illustration dynamique de programmes préexistants ou la création de "films" pédagogiques. Le principal intérêt résidant alors dans la simplicité de mise en oeuvre et dans l'importance des possibilités d'interaction.

En ce qui concerne les perspectives ultérieures, nous citerons les projets suivants:

- définition de mécanismes d'entrées graphiques analogues aux mécanismes de sorties. Ceci permettant, de la même manière, de définir la forme des entrées extérieurement au programme d'application (clavier alphanumérique, fichiers, dispositifs graphiques...).

- étude de mécanismes de dialogue homme-machine, permettant de définir simplement un dialogue dans un programme d'application.

- amélioration de certains mécanismes afin d'obtenir une plus grande efficacité ou une plus grande puissance.

ANNEXES

A N N E X E 1

Illustration d'un programme de recherche de l'élément maximum dans un tableau, à l'aide d'un système graphique général (GSP) puis d'un système graphique "plus proche" de l'utilisateur (GRIGRI) et enfin du système graphique d'interprétation des données.

1/ Programme initial écrit en FORTRAN

```
        integer Tab (20), max, N
        max = 0
        read (5,1) N
1       format (I2)
        do 2 k = 1,N
2       read (5,1) TAB(k)
        do 10 k = 1,N
10      write (6,11) Tab(k)
11      format ('Tab(x) = ',14)
        write (6,12) Max
12      format ('Max=',14)
        do 4 k = 1,N
        if (Tab(k).GT.max) max = Tab(k)
        write (6,13) k, Tab(k), max
13      format ('k=', 14, 'Tab(k) = ', 14','max=',14
4       continue
        stop
        end
```

ordre de sorties imprimantes

ordre de sortie imprimante

2/ Programme écrit en FORTRAN avec GSP

```
        integer Tab(20), max,N
        max = 0
        read (5,1)N
1       format (I2)
        do 2 k = 1,N
2       read (5,1) Tab(k)
        NL(1) = 5
        call ingsp (NG, NL)
        call indev (NG, 10, NØ)
        call cratl (NØ, NI)
        call mlits (NI, 3)
        call mlpeo (NI, 2, 1)
```

initialisation graphique

```
      call ingds (NØ, NF)
      call ingds (NØ, NF2)
      call sgram (NF, 2)
      call sgram (NF2, 2)
      call sdatm (NF, 1, 1)
      call sdatm (NF2, 1, 1)
      call scham (NF, 3)
      call scham (NF2, 3)
      call sdatl (NF, 0., 0., 100., 100.)
      call sdatl (NF2, 0., 0., 100., 100.)

      real X(6), Y(6), H, XO, YO, X1, Y1
      H = 60/N
      XO = 20.
      YO = 20.
      do 3f = 1,N
      X(1) = XO
      Y(1) = YO
      X(2) = X(1)+10
      Y(2) = Y(1)
      X(3) = X(2)
      Y(3) = Y(2)+H
      X(4) = X(1)
      Y(4) = Y(3)
      X(5) = X(1)
      Y(5) = Y(1)
      call stpos (NF, X(1), Y(11))
      call pline (NF, X, Y, NL, NL, NL, 4)
      call benv (Tab(k), IS, 103, 4)
      X1 = XO+3
      Y1 = YO+(H/2)
3     YO = YO+H
      call Ptext (NF, 'TAB', 3, NL, NL, NL, 23., 18.)

      X(1) = 70.
      Y(1) = 50.
      X(2) = X(1)+10
      Y(2) = Y(1)
      X(3) = X(2)
      Y(3) = Y(2)+5
      X(4) = X(1)
```

génération de la liste d'affichage pour la représentation du tableau 'TAB'

génération de la liste d'affichage pour la représentation de l'entier 'MAX'

```
Y(4) = Y(3)
Y(5) = X(1)
Y(5) = Y(1)
call stpos (NF, X(1), Y(1))
call pline (NF, X, Y, NL, NL, NL, 4)
call Ptext (NF, 'MAX', 3, NL, NL, NL, 73., 48.)
call exec (NF)
do 4 k = 1,N
if (Tab (k).GT.max) max = tab(k)

call  bcnv (max, IS, 103, 4)
call Ptext (NF2, IS, 4, NL, NL, NL, 73., 53.)
YO = H * (K-1)+20.
X(1) = XO+10
Y(1) = YO+(H/2)
Y(2) = X(1)+2
Y(2) = YO
X(3) = X(2)+4
Y(3) = Y(2)
X(4) = X(3)
Y(4) = Y(3)+H
X(5) = X(2)
Y(5) = Y(4)
X(6) = X(1)
Y(6) = Y(1)
call stpos (NF2, X(1), Y(1))
call pline (NF2, X, Y , NL, NL, NL, 5)
X1 = XO+12
Y1 = YO+(H/2)
call Ptext (NF2, 'I', 1, NL, NL, NL, X1, Y1)
call exec (NF2)

call enatn (NI,1)
call rqatn (NI, I, 2, NL, 1)
call dsatn (NI, 1)
call reset (NF2)
call reset (NF)
call tmgds (NF2)
call tmgds (NF)
call tmdev (NØ)
call tmgsp (NG)
stop
end
```

génération de la liste d'affichage pour la représentation de la valeur de l'entier 'MAX' et de l'index 'K'

attente

3) Programme écrit en FORTRAN avec GRIGRI

```
      integer Tab (20), max, N
      max = 0
      read (5,1) N
1     format (I2)
      do 2 k = 1, N
2     read (5,1) Tab (k)

      call figure (1, 1, 0., 0., 100., 100.)
      call figure (2, 1, 0., 0., 100., 100.)

      real X(6), Y(6), H, XO, YO, X1, Y1
      H = 60/N
      XO = 20.
      YO = 20.
      do 3 k = 1,N
      X(1) = XO
      Y(1) = YO
      X(2) = X(1)+10
      Y(2) = Y(1)
      X(3) = X(2)
      Y(3) = Y(2)+H
      X(4) = X(1)
      Y(4) = Y(3)
      X(5) = X(1)
      Y(5) = Y(1)
      call affich (1, 1, 4, X, Y)
      X1 = XO+3
      Y1 = YO+(H/2)
      call editer (1, 1, 'Z3', X1, Y1)
      call edenti (1, 1, 1, tab (k))
3     YO = YO+H
      call editer (1, 1, '/TAB/', 23., 18.)

      X(1) = 70.
      Y(1) = 50.
      X(2) = X(1)+10
      Y(2) = Y(1)
      X(3) = X(2)
      Y(3) = Y(2)+5
      X(4) = X(1)
      Y(4) = Y(3)
```

initialisation graphique

génération de la liste d'affichage pour la représentation du tableau'TAB'

```
      X(5) = X(1)
      Y(5) = Y(1)
      call affich (1, 1, 4, X, Y)
      call editer (1, 1, '/MAX/', 73., 48.)
      call editer (2, 1,'Z3', 73., 53.)
      do 4 k = 1,N
      if (Tab(k).GT.max) max = Tab(k)
      call edenti (2, 1, 1, max)
      YO = H * (k-1)+20.
      X(1) = XO+10
      Y(1) = YO+(H/2)
      X(2) = X(1)+2
      Y(2) = YO
      X(3) = X(2)+4
      Y(3) = Y(2)
      X(4) = X(3)
      Y(4) = Y(3)+H
      X(5) = X(2)
      Y(5) = Y(4)
      X(6) = X(1)
      Y(6) = Y(1)
      call affich (2, 1, 5, X, Y)
      X1 = XO+12
      Y1 = YO+(H/2)
      call editer (2, 1, '/K/', X1, Y1)

      call menu ('1 ')
4     call efface (2)
      call efface (1)
      stop
      end
```

génération de la liste d'affichage pour la représentation de l'entier 'MAX'

génération de la liste d'affichage pour la représentation de la valeur de l'entier 'MAX' et de l'index 'K'

attente

4/ Programme écrit en FORTRAN avec le système d'interprétation des données

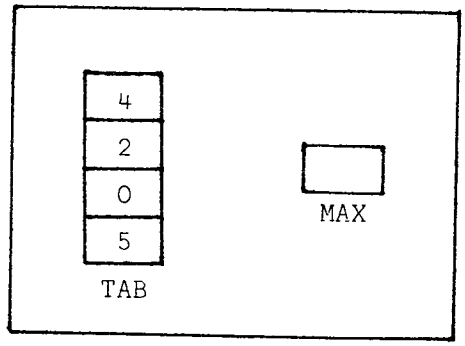
```
      integer Tab (20), max, N
      max = 0
      read (5,1) N
1     format (I2)
      do 2 k = 1,N
2     read (5,1) tab (k)
      ecran (1, tab (1:N), max)
      afficher (.true.)
```

ordre de sortie graphique

```
      do 4 k = 1,N
      if (Tab(k).GT.max)max = tab (k)
      ecran (2, tab(1:N), max, k)                    ]
4     afficher (.true.)                              ] ordre de sortie graphique
      stop
      end
```

Les descripteurs utilisés étant alors des descripteurs standard.

Ce qui donne dans les trois cas, la représentation suivante (pour un tableau de 4 éléments par exemple):



première image (après lecture du tableau)

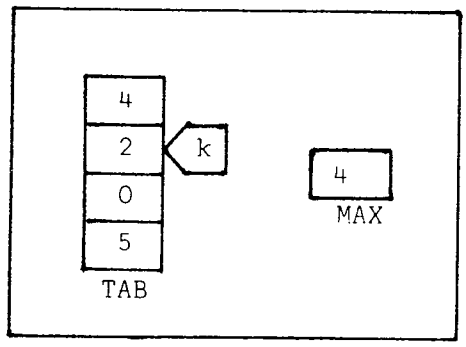


image qui correspond au deuxième pas de l'itération

ANNEXE 2

Grammaire d'un langage de description graphique avec inclusions sémantiques

<desc>::= descripteur nombre 'f1' : <ldcl><ug><listeug> fin 'f36'.

<ldcl>::= <dcl>; <ldcl> | Ø

<dcl> ::= externe (nombre 'f2') ident 'f3' <lident> | interne 'f35'ident 'f3' <lident>

<lident> ::= , ident 'f3'<lident>|Ø

<listeug> ::= ; <ug><listeug>| Ø

<ug> ::= <objet isolé><param> |<figure> 'f30' <param> | iter <nb> : <ug><listeug> fin 'f6'

<nb> ::= nombre 'f4' | ×'f5'

<objet isolé> := <obstat>|<obdyn1>|<obdyn2><ident>'f11' | texte 'f12' <B>

<B> ::= (nombre 'f32') | Ø

<figure> ::= <fig>|ident 'f13' := <fig>

<fig> ::= <figstat>|<figdyn>

<obstat> ::= cell 'f14' (<Base> nombre 'f15' 'f31' <A1>)

<Base> ::= E 'f16',| L 'f17',| Ø

<A1> ::= , nombre f32' | Ø

<obdyn 1> ::= cellvar 'f18' (<Base><code>, nombre 'f15')

<code> ::= H 'f19' | S 'f20' | Z 'f21'

<obdyn 2>::= index (<type> 'f22') | pointeur 'f23'

<type> ::= 1| 2| 3|

<figstat> ::= Pile 'f24' (nombre 'f15', <objet>)| Vecteur 'f25' (<dim1><objet>)|
matrice 'f26 (<dim2><objet>)

<objet> ::= <obstat>|<obdyn1>

<dim1> ::= nombre 'f32',| Ø

<dim2> ::= nombre 'f33' × nombre 'f32',| Ø

<figdyn> ::= courbe 'f27' <A2>|Graphe 'f28' <sommet>

<A2> ::= (<A3>)| Ø

<A3> ::= nombre 'f33' <A4>|axes 'f29'

<A4> ::= ,axes 'f29'| Ø

<sommet> ::= (f'34' <obstat>)|Ø

<param> ::= ø<B1> ø|Ø

<B1> ::= T = nombre 'f7' × nombre 'f8' <B2>|<B3>

<B2> ::= , <B3>|Ø

<B3> ::= p = nombre 'f9' × nombre 'f10'

Les points d'extension de la grammaire sont désignés par les métasymboles suivants:
<obstat>, <obdyn1>, <obdyn2>, <figstat>, <figdyn>

Les fonctions sémantiques permettent de coder sous forme interne la description analysée, de ranger les paramètres effectifs et d'établir les divers chaînages. En ce qui concerne les extensions, il existe un ensemble de fonctions sémantiques que le système incorpore dans les nouvelles règles afin de traiter correctement le type de l'extension et la récupération des paramètres effectifs.

ANNEXE 3

Une syntaxe proposée pour les ordres de communication "programme d'application - système d'interprétation"

1/ Ordre "Ecran"

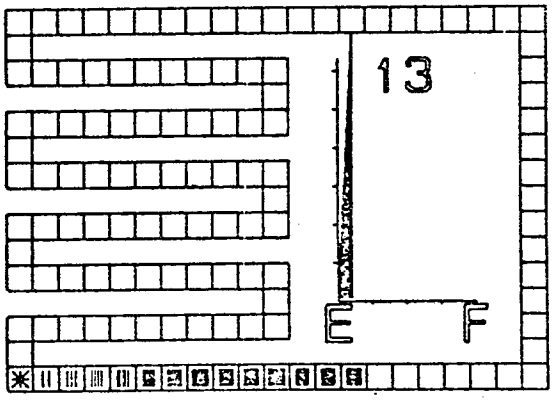
<transfert> ::= ecran (<numero>, <liste param>)
<liste param> ::= <param>, <listparam>|<param>
<param> ::= <nom><item>|<nom>§<item>|<nom>(<liste param>)
<nom> ::= /<ident>/| Ø
<item> ::= <identificateur de variable simple>|<identificateur de tableau>|
<identificateur de tableau>(<liste bornes>)|
<expression>
<liste bornes> ::= <borne>, <liste bornes>|<borne>
<borne> ::= <borne simple>|<dim>
<borne simple> ::= <nombre>|<expression entière>
<dim> ::= <borne simple>:<borne simple>|x
<numéro> ::= <expression entière>

Les termes "identificateur", "nombre", "expression" et "expression entière" ont alors la même signification que dans le langage de programmation employé.

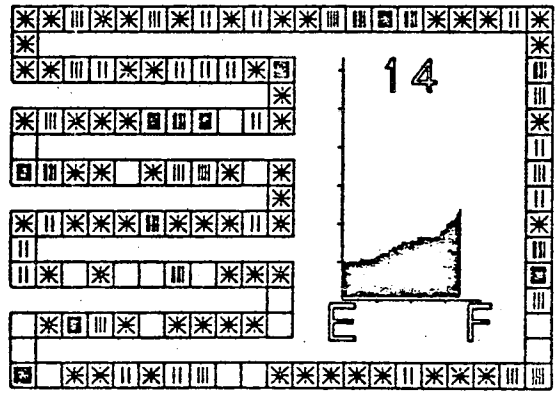
2/ Ordre "Affiche"

<affichage> ::= affiche <param>
<param> ::= (<numéro>) | (<numéro>, <bool>) | (<bool>) | Ø
<numéro> ::= <expression entière>
<bool> ::= <expression booléenne>

ANNEXE 4



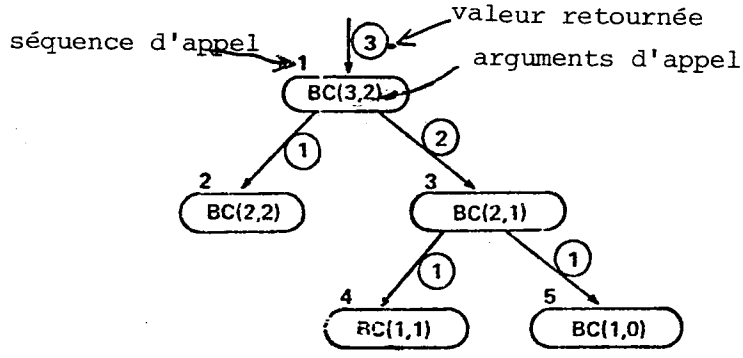
A single entry point



Quadratic Hash - Random Keys, 110 entries

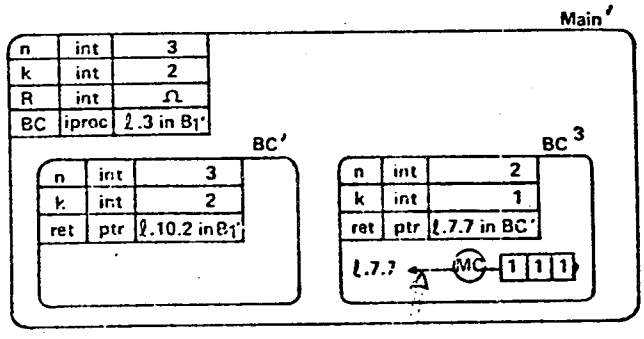
Représentation des tables de Hash-code par Hopgood [3]

```
  0 1 2 3 4 5 6 7 8 9
                    Main
1  begin
2    integer   n,k,R;
                        BC
3    integer   procedure   BC(n,k) ;
4      value     n, k;  integer     n, k;
5      if  K=0 V n=k   then BC:  = 1
6      else   BC: = BC (n-1,k)
7                        +BC (n -1, k -1)
8    ;
9    read  (n, k);
10   R: =BC  (n,k);
11   print  (n, k, R)
12   end
```



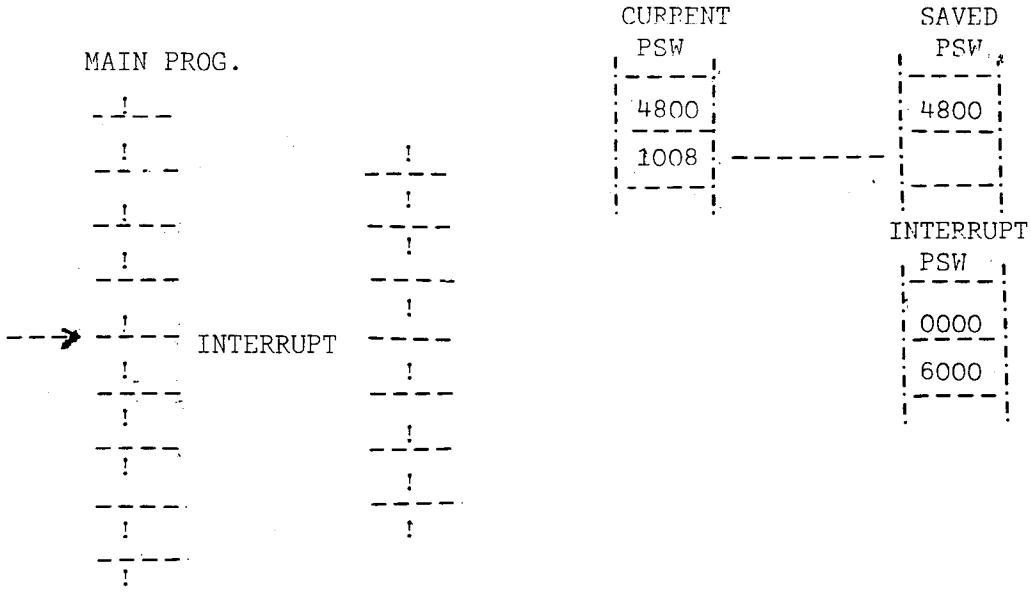
Arbre des appels récursifs

Représentation du programme

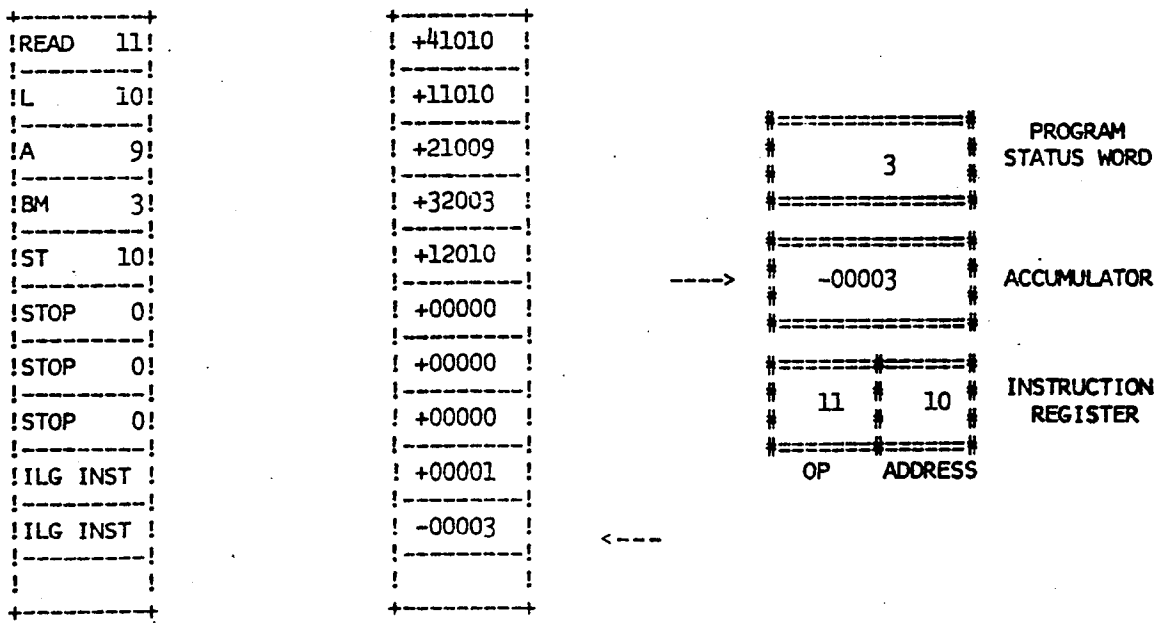


Instantanée de l'état du programme (avant le retour du 3éme appel)

Représentation de la sémantique des programmes [5]



Démonstration du mécanisme des interruptions [P4]



Démonstration du fonctionnement d'une machine de base [P4]

BIBLIOGRAPHIE

BIBLIOGRAPHIE

G1 M.LUCAS
"Technologie, programmation et utilisation des consoles de visualisation (état des recherches actuelles)"
Rapport DRME, mai 1974.

G2 M.LUCAS
"L'exploitation et l'organisation des systèmes graphiques"
Compte rendu du groupe de travail AFCET, journées graphiques AFCET IRIA, décembre 1973.

G3 M.LUCAS
"Les systèmes de visualisation graphique, état des recherches actuelles"
Séminaire de programmation, Université scientifique et médicale de GRENOBLE.

G4 P.MORVAN, M.LUCAS
"Images et ordinateur: introduction à l'infographie interactive"
Sciences humaines et sociales - Service informatique LAROUSSE, septembre 1976.

S1 "Graphic subroutine package"
IBM Systems reference library.

S2 C.LAUGIER, A.LEDUC LEBALLEUR, M.LUCAS, F.MARTINEZ
"GRIGRI: logiciel de base pour l'utilisation des consoles de visualisation du CICG"
Equipe graphique, novembre 1975.

S3 E.SALTEL
"Utilisation de la génération dynamique d'images et des processeurs graphiques dans le software metavisu III"
Thèse de 3ème cycle, PARIS, mai 1974.

S4 H.E.KULSRUD
"A general purpose graphic language"
Comm.ass.Comput. - march, vol.11, n°4, 1968

S5 R.A.MORRISON
"Graphic language translation with a language independant processor"
1967 Fall joint computer conf. AFIPS
Conf. Proc. WASHINGTON D.C., Thompson.

S6 P.D.ROWNER, D.A.HENDERSON
"On the implementation of ambit/6: a graphical programming language"
in Proc.1969 Int.joint Conf. artificial intelligence, 1969

S7 P.A.WOODSFORD
"The design and implementation of the GINO 3D graphics
software package"
Software practice an experience, vol.1, p.335, oct.1971.

S8 W.A.NEUMAN, R.F.SPROULL
"An approach to graphics system design"
Proceedings of the IEEE, vol.62, n°4, p.471/483, april 1974.

S9 J.MICHENER, I.COTTON, K.KEKKEY, D.LIDDLE, E.MEYER
"Graphics protocol"
RFC 493, avril 1973.

S10 R.F.SPROULL, E.L.THOMAS
"A network graphics protocol"
Projet MAC, août 1974.

L1 O.J.DAHL, B.MYHRHANG, K.NYGAARD
"The SIMULA 67 common base language"
Norwegian computing centre, TORSKNINGSVEIEN 1B, OSLO, 1968

L2 O.J.DAHL, K.NYGAARD
"SIMULA: an algol based simulation language"
Comm.ACM 9, 9, pp.671/678, 1966

L3 M.GRIFFITHS
"Analyse synthaxique pour la production de compilateurs"
U.S.M.G. octobre 1973.

L4 R.W.FLOYD
"Syntactic analysis and operator precedence"
Journal of the ACM, vol.10, N°3, p.316, 1963.

L5 J.D.ICHBIOH, S.P.MORSE
"A technique for generating almost optimal floyd evans productions for precedence grammars"
Communications of the ACM, vol.13, n°8, august 1970.

L6 R.A.BROOKER, I.R.MACCALLUM, D.MORRIS, J.S.ROHL
"The compiler compiler"
Annual review in automatic programming 3, p.229, 1963.

L7 J.LUCAS, K.WALK
"On the formal definition of PL/1"
Annual review in automatic programming 6, p.105, 1971.

E1 Ph.JORRAND
"Contribution au développement des langages extensibles"
Thèse d'état - U.S.M.G. janvier 1975

E2 PH.JORRAND, D.BERT
"On some basic concepts for extensible programming languages"
International computing symposium, pp.2/16, VENICE, april 1972.

E3 D.BERT
"Etude d'éléments fondamentaux des langages de programmation"
Thèse de 3ème cycle, U.S.M.G, mai 1973.

E4 N.WIRTH, H.WEBER
"EULTR: generalization of algol and its formal definition"
Communication of the ACM, january and february 1966, vol.9,N°1, pp 13/23 N°2, pp.89/99.

P1 F.R.A.HOPGOOD
"Computer animation as a tool in teaching computing science"
Information processing 74, North Holland publishing company, 1974.

P2 P.E.DRENICK
"Computer animated algorithms"
IEEE Transactions on education, vol.E 17, mai 1974.

P3 I.ELLIOT, ORGANIK and J.W.THOMAS
"Computer generated semantics displays"
Information processing 1974, North Holland Publishing, 1974.

P4 D.R.LEVINE
"Computer controlled display demonstrations of dynamic concepts in computer science"
Technical report ≠ 37.

P5 W.A.MARTIN
"Sorting"
Computing surveys, vol.3, N°4, décembre 1971.

P6 G.VEILLON
"Algorithmes combinatoires"
USMG, GRENOBLE.

P7 A.ELLIS
"The use and miuse of computers in education"
McGraw Hill, N.Y. 1974

P8 J.B.JOHNSTON
"The contour model of block structured processes"
Proceedings of ACM symposium on data structures in programming languages. SIGPLAN Notices, feb.1971, pp.35/82.

P9 V.R.BASILI, A.J.TURNER
"A hierarchical machine model for the semantics of programming languages"
Proceedings of ACM. IEEE symposium on high level language computer architecture, nov.1973, pp.152/164.

P10 W.TRACZ
"The use of ATOPSS for presenting elementary operating system concepts"
SIGCSE Bulletin, vol.6, pp 74/78, feb.1974.

P11 S.J.K.McADAMS, A.R.DEKOK
"Computer graphics as an aid to teaching geometric transformations"
SIGCSE Bulletin, vol.8, pp 137/143, feb.1976.

P12 R.CHENG
"On line large screen display system for computer instruction"
SIGCSE Bulletin, vol.8, pp.179/181, feb.1976.

P13 C.WU
"CAI tutorial method of teaching thermodynamics at US NAVAL ACADEMY"
SIGCSE Bulletin, vol.6, pp 97/100, feb.1974.

P14 E.L.LAMIE
"Using GPSS to teach operating systems concepts"
SIGCSE Bulletin, vol.8, pp.174/178, feb.1976.

P15 W.C.DOWLING
"A computer graphics course for undergraduate engineers"
SIGCSE bulletin, vol.6, pp5/8, june 1974.

P16 S.C.SHAPIRO, D.P.WITMER
"Interactive visual simulators for begining programming students"
SIGCSE bulletin, vol.6, pp.11/14, feb.1974.

P17 S.BELL, E.J.GILBERT
"Learning recursion with syntax diagrams"
SIGCSE bulletin, vol.6, sept.1974.

I1 J.AZEMA, J.C.SAILLARD
"Visualisation interactive de graphes"
Rapport contrat DRME - Laboratoire d'informatique USMG - mars 1973

I2 C.LAUGIER, F.MARTINEZ
"Visualisation interactive de graphes"
Rapport contrat DRME, Labo d'informatique USMG, dec.1974, juillet 1975

I3 C.LAUGIER
"Illustration dynamique de programmes à l'aide d'une console de visualisation"
Séminaire de programmation - USMG - mars 1976.

14 C.LAUGIER
"Illustration dynamique de programmes à l'aide d'une console de visualisation"
Congrés AFCET 1976/TTI: Panorama de la nouveauté informatique en France
Novembre 1976.