Un éditeur graphique pour le système CASCADE : EDICAS

Jean-Charles Marty

THESE

présentée à

l'Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR DE 3ème CYCLE
«Informatique»

par

Jean-Charles MARTY

UN EDITEUR GRAPHIQUE POUR LE SYSTEME *CASCADE: EDICAS*

Thèse soutenue le 22 octobre 1984 devant la commission d'examen.

G. VEILLON	Président
D. BORRIONE	
J. MERMET	Examinateurs
P. PRINETTO	
M. SCHLUMBERGER	

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : D. BLOCH

Vice-Président : René CARRE
Hervé CHERADAME
Marcel IVANES

PROFESSEURS DES UNIVERSITES :

ANCEAU François	E.N.S.I.M.A.G.
BARRAUD Alain	E.N.S.I.E.G.
BAUDELET Bernard	E.N.S.I.E.G.
BESSON Jean	E.N.S.E.E.G.
BLIMAN Samuel	E.N.S.E.R.G.
BLOCH Daniel	E.N.S.I.E.G.
BOIS Philippe	E.N.S.H.G.
BONNETAIN Lucien	E.N.S.E.E.G.
BONNIER Etienne	E.N.S.E.E.G.
BOUVARD Maurice	E.N.S.H.G.
BRISSONNEAU Pierre	E.N.S.I.E.G.
BUYLE BODIN Maurice	E.N.S.E.R.G.
CAVAIGNAC Jean-François	E.N.S.I.E.G.
CHARTIER Germain	E.N.S.I.E.G.
CHENEVIER Pierre	E.N.S.E.R.G.
CHERADAME Hervé	U.E.R.M.C.P.P.
CHERUY Arlette	E.N.S.I.E.G.
CHIAVERINA Jean	U.E.R.M.C.P.P.
COHEN Joseph	E.N.S.E.R.G.
COUMES André	E.N.S.E.R.G.
DURAND Francis	E.N.S.E.E.G.
DURAND Jean-Louis	E.N.S.I.E.G.
FELICI Noël	E.N.S.I.E.G.
FOULARD Claude	E.N.S.I.E.G.
GENTIL Pierre	E.N.S.E.R.G.
GUERIN Bernard	E.N.S.E.R.G.
GUYOT Pierre	E.N.S.E.E.G.
IVANES Marcel	E.N.S.I.E.G.
JAUSSAUD Pierre	E.N.S.I.E.G.
JOUBERT Jean-Claude	E.N.S.I.E.G.
JOURDAIN Geneviève	E.N.S.I.E.G.
LACOUME Jean-Louis	E.N.S.I.E.G.
LATOMBE Jean-Claude	E.N.S.I.M.A.G.

.../...

LESSIEUR Marcel	E.N.S.H.G.
LESPINARD Georges	E.N.S.H.G.
LONGEQUEUE Jean-Pierre	E.N.S.I.E.G.
MAZARE Guy	E.N.S.I.M.A.G.
MOREAU René	E.N.S.H.G.
MORET Roger	E.N.S.I.E.G.
MOSSIERE Jacques	E.N.S.I.M.A.G.
PARIAUD Jean-Charles	E.N.S.E.E.G.
PAUTHENET René	E.N.S.I.E.G.
PERRET René	E.N.S.I.E.G.
PERRET Robert	E.N.S.I.E.G.
PIAU Jean-Michel	E.N.S.H.G.
POLOUJADOFF Michel	E.N.S.I.E.G.
POUPOT Christian	E.N.S.E.R.G.
RAMEAU Jean-Jacques	E.N.S.E.E.G.
RENAUD Maurice	U.E.R.M.C.P.P.
ROBERT André	U.E.R.M.C.P.P.
ROBERT François	E.N.S.I.M.A.G.
SABONNADIERE Jean-Claude	E.N.S.I.E.G.
SAUCIER Gabrielle	E.N.S.I.M.A.G.
SCHLENKER Claire	E.N.S.I.E.G.
SCHLENKER Michel	E.N.S.I.E.G.
SERMET Pierre	E.N.S.E.R.G.
SILVY Jacques	U.E.R.M.C.P.P.
SOHM Jean-Claude	E.N.S.E.E.G.
SOUQUET Jean-Louis	E.N.S.E.E.G.
VEILLON Gérard	E.N.S.I.M.A.G.
ZADWORNY François	E.N.S.E.R.G.

PROFESSEURS ASSOCIES

BASTIN Georges	E.N.S.H.G.
BERRIL John	E.N.S.H.G.
CARREAU Pierre	E.N.S.H.G.
GANDINI Alessandro	U.E.R.M.C.P.P.
HAYASHI Hirashi	E.N.S.I.E.G.

PROFESSEURS UNIVERSITE DES SCIENCES SOCIALES (Grenoble II)

BOLLIET Louis
Chatelin Françoise

PROFESSEURS E.N.S. Mines de Saint-Etienne

RIEU Jean
SOUSTELLE Michel

CHERCHEURS DU C.N.R.S.

ERUCHART Robert	Directeur de Recherche
VACHAUD Georges	Directeur de Recherche

.../...

ALLIBERT Michel	Maître de Recherche
ANSARA Ibrahim	Maître de Recherche
ARMAND Michel	Maître de Recherche
BINDER Gilbert	
CARRE René	Maître de Recherche
DAVID René	Maître de Recherche
DEPORTES Jacques	
DRIOLE Jean	Maître de Recherche
GIGNOUX Damien	
GIVORD Dominique	
GUELIN Pierre	
HOPFINGER Emil	Maître de Recherche
JOUD Jean-Charles	Maître de Recherche
KAMARINOS Georges	Maître de Recherche
KLEITZ Michel	Maître de Recherche
LANDAU Ioan-Dore	Maître de Recherche
LASJAUNIAS J.C.	
MERMET Jean	Maître de Recherche
MUNIER Jacques	Maître de Recherche
PIAU Monique	
PORTESEIL Jean-Louis	
THOLENCE Jean-Louis	
VERDILLON André	

CHERCHEURS du MINISTERE de la RECHERCHE et de la TECHNOLOGIRE (Directeurs et Maîtres de Recherches, ENS Mines de St. Etienne)

LESBATS Pierre	Directeur de Recherche
BISCONDI Michel	Maître de Recherche
KOBYLANSKI André	Maître de Recherche
LE COZE Jean	Maître de Recherche
LALAUZE René	Maître de Recherche
LANCELOT Francis	Maître de Recherche
THEVENOT François	Maître de Recherche
TRAN MINH Canh	Maître de Recherche

PERSONNALITES HABILITEES à DIRIGER des TRAVAUX de RECHERCHE (Décision du Conseil Scientifique)

ALLIBERT Colette	E.N.S.E.E.G.
BERNARD Claude	E.N.S.E.E.G.
BONNET Rolland	E.N.S.E.E.G.
CAILLET Marcel	E.N.S.E.E.G.
CHATILLON Catherine	E.N.S.E.E.G.
CHATILLON Christian	E.N.S.E.E.G.
COULON Michel	E.N.S.E.E.G.
DIARD Jean-Paul	E.N.S.E.E.G.
EUSTAPOPOULOS Nicolas	E.N.S.E.E.G.
FOSTER Panayotis	E.N.S.E.E.G.

.../...

GALERIE Alain	E.N.S.E.E.G.
HAMMOU Abdelkader	E.N.S.E.E.G.
MALMEJAC Yves	E.N.S.E.E.G. (CENG)
MARTIN GARIN Régina	E.N.S.E.E.G.
NGUYEN TRUONG Bernadette	E.N.S.E.E.G.
RAVAINE Denis	E.N.S.E.E.G.
SAINFORT	E.N.S.E.E.G. (CENG)
SARRAZIN Pierre	E.N.S.E.E.G.
SIMON Jean-Paul	E.N.S.E.E.G.
TOUZAIN Philippe	E.N.S.E.E.G.
URBAIN Georges	E.N.S.E.E.G. (Laboratoire des ultra-réfractaires ODEILLON)
GUILHOT Bernard	E.N.S. Mines Saint Etienne
THOMAS Gérard	E.N.S. Mines Saint Etienne
DRIVER Julien	E.N.S. Mines Saint Etienne
BARIBAUD Michel	E.N.S.E.R.G.
BOREL Joseph	E.N.S.E.R.G.
CHOVET Alain	E.N.S.E.R.G.
CHEHIKIAN Alain	E.N.S.E.R.G.
DOLMAZON Jean-Marc	E.N.S.E.R.G.
HERAULT Jeanny	E.N.S.E.R.G.
MONLLOR Christian	E.N.S.E.R.G.
BORNARD Guy	E.N.S.I.E.G.
DESCHIZEAU Pierre	E.N.S.I.E.G.
GLANGEAUD François	E.N.S.I.E.G.
KOFMAN Walter	E.N.S.I.E.G.
LEJEUNE Gérard	E.N.S.I.E.G.
MAZUER Jean	E.N.S.I.E.G.
PERARD Jacques	E.N.S.I.E.G.
REINISCH Raymond	E.N.S.I.E.G.
ALEMANY Antoine	E.N.S.H.G.
BOIS Daniel	E.N.S.H.G.
DARVE Félix	E.N.S.H.G.
MICHEL Jean-Marie	E.N.S.H.G.
OBLED Charles	E.N.S.H.G.
ROWE Alain	E.N.S.H.G.
VAUCLIN Michel	E.N.S.H.G.
WACK Bernard	E.N.S.H.G.
BERT Didier	E.N.S.I.M.A.G.
CALMET Jacques	E.N.S.I.M.A.G.
COURTIN Jacques	E.N.S.I.M.A.G.
COURTOIS Bernard	E.N.S.I.M.A.G.
DELLA DORA Jean	E.N.S.I.M.A.G.
FONLUPT Jean	E.N.S.I.M.A.G.
SIFAKIS Joseph	E.N.S.I.M.A.G.
CHARUEL Robert	U.E.R.M.C.P.P.
CADET Jean	C.E.N.G.
COEURE Philippe	C.E.N.G. (LETI)

.../...

DELHAYE Jean-Marc	C.E.N.G. (STT)
DUPUY Michel	C.E.N.G. (LETI)
JOUVE Hubert	C.E.N.G. (LETI)
NICOLAU Yvan	C.E.N.G. (LETI)
NIFENECKER Hervé	C.E.N.G.
PERROUD Paul	C.E.N.G.
PEUZIN Jean-Claude	C.E.N.G. (LETI)
TAIEB Maurice	C.E.N.G.
VINCENDON Marc	C.E.N.G.

LABORATOIRES EXTERIEURS

DEMOULIN Eric	C.N.E.T.
DEVINE	C.N.E.T. (R.A.B.)
GERBER Roland	C.N.E.T.
MERCKEL Gérard	C.N.E.T.
PAULEAU Yves	C.N.E.T.
GAUBERT C.	I.N.S.A. Lyon

Je tiens particulièrement à remercier ici:

Monsieur Gérard VEILLON, professeur à l'Institut National Polytechnique de Grenoble, pour m'avoir fait l'honneur de présider le jury de cette thèse. Je tiens à lui exprimer ma sincère reconnaissance.

Monsieur Jean MERMET, maître de recherches au C.N.R.S., pour avoir dirigé cette thèse et pour les critiques et les encouragements prodigués tout au long de mon travail.

Madame Dominique BORRIONE, professeur à l'université d'Aix-Marseille, pour le temps qu'elle a investi dans mon projet (y compris pendant ses vacances). De nombreux résultats exposés ici sont le fruit d'une étroite collaboration avec elle.

Monsieur Paolo PRINETTO, professeur assistant à l'Institut Polytechnique de Turin, pour avoir accepté de juger mon travail.

Monsieur Maurice SCHLUMBERGER, responsable du centre de recherche de de Grenoble à CAP SOGETI INNOVATION, pour les nombreuses discussions que nous avons eues.

La compagnie CAP SOGETI INNOVATION et l'A.N.R.T. sans lesquelles ces études n'auraient pas eu lieu.

Je tiens également à remercier tous ceux qui ont contribué aux travaux rapportés ici par leurs critiques ou suggestions et en particulier B. DAVID, Y. DURAND, JF. GRABOWIECKI, M.T. GUICHARD, S. LAFONT, C. LE FAOU, J. ROUTIN, P. UVIETTA.

Je remercie également Francette BONNABAUD qui a dactylographié avec soin cet ouvrage et le service de reprographie de l'I.M.A.G. pour la qualité du tirage effectué.

Enfin, je remercie tous ceux qui m'ont aidé par leurs encouragements durant ces deux années. Je pense particulièrement à ma famille et à mon épouse, Annick, pour le réconfort qu'elle m'a toujours apporté et la patience dont elle a fait preuve.

PLAN

INTRODUCTION

L'objet de cette thèse est d'exposer un éditeur graphique que nous avons réalisé pour la description de circuits. Cet éditeur est inclus dans un système beaucoup plus important : le système CASCADE (Conception Assistée de Systèmes et Circuits Analogiques et Digitaux Electroniques) (Mer83), (Bor82).

Origine du système CASCADE

M. Feuer écrivait récemment (Feuer83): "The field of V.S.L.I. design automation has grown so large in the last twenty years that a complete treatment would require an encyclopedia".

Il n'est donc pas dans notre intention de faire ici un tour d'horizon complet des travaux accomplis dans ce domaine. Nous remarquons simplement qu'en général un langage de description est attaché à un niveau particulier de description (Lip77). Ce niveau de description peut varier depuis le niveau "architecture système" jusqu'au niveau "électrique fin".

Une classification de différents systèmes permettant de décrire des circuits est déjà proposée dans (Mer83); elle illustre bien cette spécificité des différents langages de description.

Pour permettre la simulation et la validation de gros systèmes (et des sous-systèmes qui ne sont pas décrits au même niveau de langage), il est nécessaire d'unifier la méthode de conception et de permettre des traitements multiniveaux.

Le projet international CONLAN (PBB83), (PBB80), (PiB82), (Bor81) avait cet objectif. Une réflexion théorique a abouti permettant la cohabitation de tous les niveaux logiques.

Le projet CASCADE a pour objectif de mettre en oeuvre de telles notions en intégrant en plus les niveaux électriques. Il fait l'objet d'un projet européen dont les partenaires principaux sont l'IMAG-MICADO (France) (maître d'oeuvre), S.A. PHILIPS I.C. (France). S.G.S. (Italie), I.M.C. (Angleterre), R.I.C. (France), P.I.I. (Pays-Bas). Ce projet a commencé le 1er février 1983 et se terminera le 1er août 1985.

Le système CASCADE

Dans le système CASCADE, toutes les descriptions sont effectuées grâce au langage CASCADE. La principale caractéristique de ce langage est la modularité des descriptions. Ainsi, lorsque l'on veut simuler un circuit quelconque, on compile séparément toutes les sous-parties du circuit (qui sont elles-mêmes des descriptions indépendantes et simulables). Ceci constitue la première phase d'analyse. Puis, on construit le modèle complet à partir des différentes descriptions compilées. Le modèle est restructuré pour rendre la simulation plus efficace. On obtient alors le modèle simulable. Ceci peut se résumer par le schéma suivant :

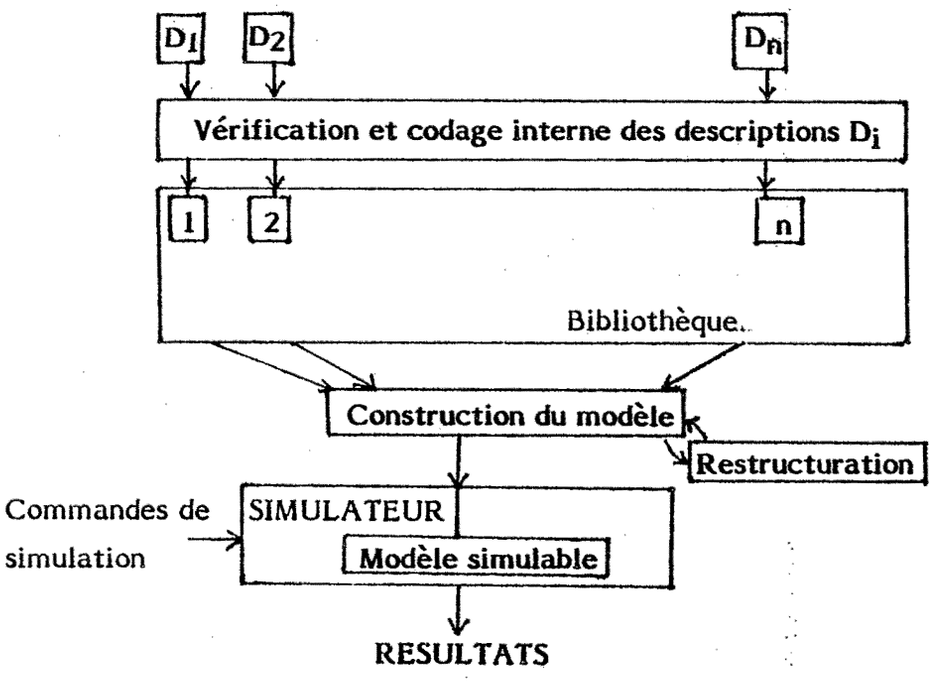


Fig1

L'éditeur graphique de CASCADE

Le langage CASCADE permet au concepteur de décrire textuellement ses circuits afin de pouvoir les simuler. Cependant, le fait d'utiliser un langage textuel pour cela peut paraître fastidieux au concepteur souvent habitué à raisonner en termes de schémas.

Créer une entrée graphique pour le système CASCADE s'est donc révélé indispensable.

L'éditeur graphique que nous décrivons ici est en fait beaucoup plus qu'un éditeur de schémas classique. En effet, il permet de générer automatiquement la description textuelle en langage CASCADE correspondant au circuit entré graphiquement (D'i--->Di sur Fig. 2). De plus, cet éditeur permet également

de décrire des circuits à tous les niveaux. Ceci pose de nombreux problèmes notamment lorsque le concepteur doit décrire plus que la structure de son circuit. En effet, entrer une structure graphiquement est relativement facile. Par contre, entrer le comportement d'un circuit graphiquement reste un problème ouvert.

Par rapport à Fig. 1, l'éditeur graphique se situe en amont du système:

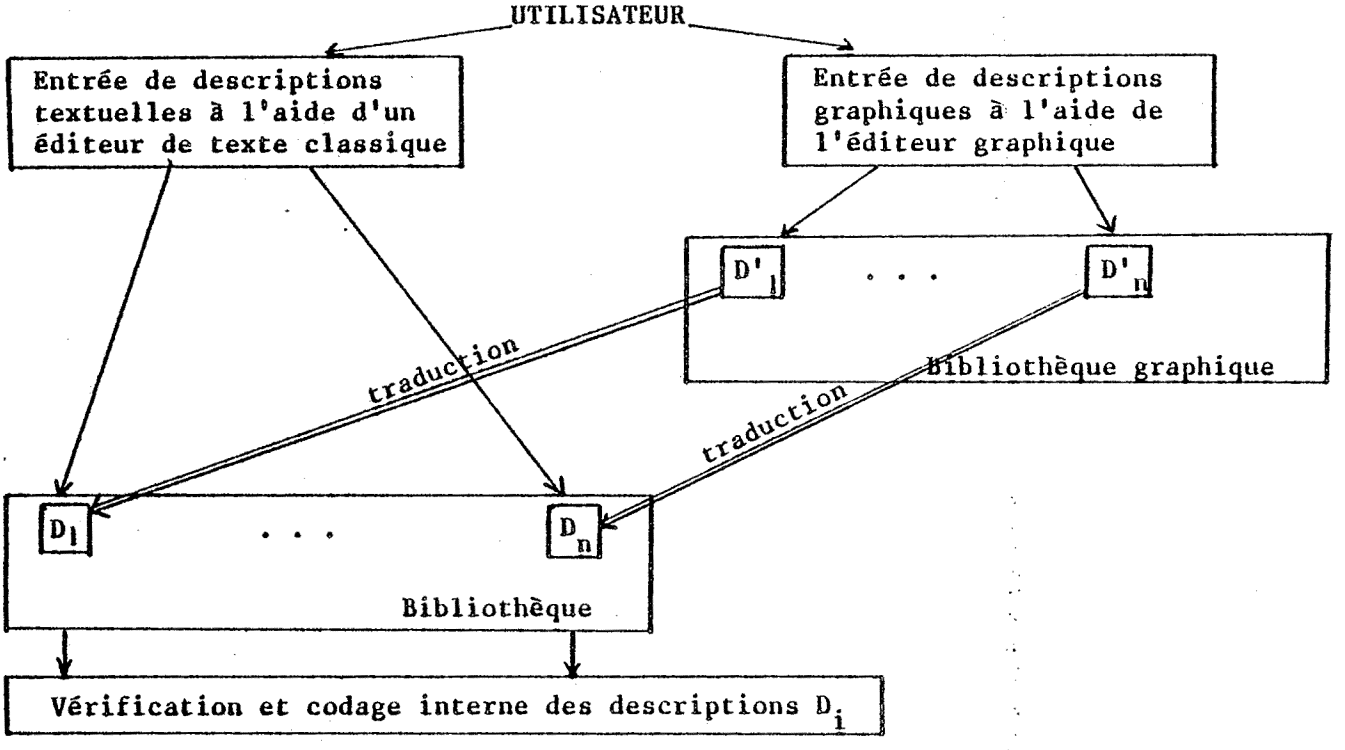


Fig 2

Nous proposons également une solution qui permet la traduction dans le sens inverse: Comment à partir d'une description textuelle obtenir une description graphique?

La solution de ce problème peut par exemple, servir à visualiser les résultats qui seront fournis par la simulation. Dans ce cas-là, c'est en aval du système que la représentation graphique sera utilisée. Il est évident que, pour le concepteur, cette simulation avec un langage de commandes graphique serait beaucoup plus "agréable" que des résultats textuels.

Ce sont tous ces aspects graphiques qui sont exposés dans cette thèse.

Nous donnons, tout d'abord, quelques notions graphiques indispensables au lecteur novice dans ce domaine (Chapitre I). Puis, nous étudions l'éditeur graphique de CASCADE, EDICAS, d'abord en ne considérant que les descriptions structurelles (Chapitre II), puis en tenant compte des descriptions qui nécessitent l'entrée d'une partie comportementale (Chapitre III). Enfin, nous tenterons de considérer le problème traité de manière plus générale en proposant une unification de certains éditeurs (Chapitre IV).

CHAPITRE I

GENERALITES SUR LE GRAPHIQUE

I.1 Les logiciels graphiques de base

I.1.1 Notions générales

I.1.2 Services attendus d'un logiciel graphique de base

I.2 Etude de quelques logiciels de base

I.2.1 Classification des primitives

I.2.2 Le logiciel GRIGRI

I.2.3 Le logiciel I.G.L

I.2.4 G.K.S.

I.2.5 Le logiciel CLOVIS

I.3 Choix d'un logiciel graphique de base pour l'éditeur graphique de CASCADE

CHAPITRE I

GENERALITES SUR LE GRAPHIQUE

Dans ce chapitre, nous exposons d'abord quelques notions fondamentales liées aux logiciels graphiques de base. Puis, nous énumérons certains de ces logiciels qui sont représentatifs de la tendance actuelle. Enfin, nous justifions le choix du logiciel graphique de base adopté pour notre application.

I.1 LES LOGICIELS GRAPHIQUES DE BASE

(Luc77), (LLM78), (EPS77), (Gue76)

Tout utilisateur désirant dessiner des objets sur un matériel donné a besoin d'un logiciel graphique de base. Nous allons définir deux notions attachées à ces logiciels:

- Leur définition et comment les utiliser, ce qui permettra de situer de tels logiciels par rapport à la machine et à l'utilisateur.

- Ce que l'utilisateur attend de ces logiciels, ce qui permet de limiter les services qu'ils doivent rendre.

I.1.1 NOTIONS GENERALES

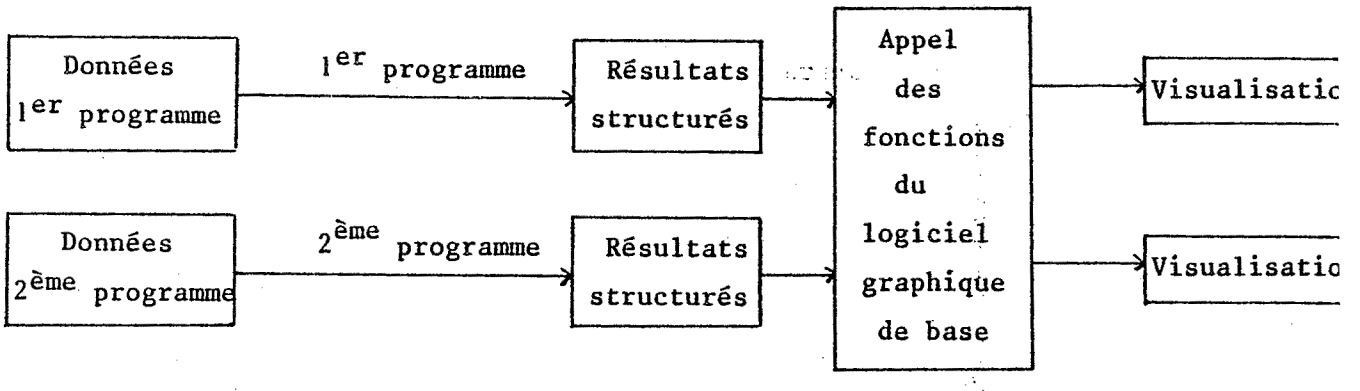
Chaque utilisateur de graphique a son problème donné, bien spécifique, qui peut aller de la gestion (tracé de courbes statistiques) jusqu'à la synthèse d'images (Mar82) en passant par la conception de nouvelles surfaces (profil de véhicules par exemple: logiciel EUCLID (The72)). Dans tous les programmes correspondant aux applications citées ci-dessus, il s'effectue un traitement de données structurées différent suivant l'application. A partir de cette structure, on peut regrouper un ensemble d'informations nécessaires pour créer le dessin correspondant. Cet ensemble est souvent appelé "scène".

En fait nous distinguons deux parties dans un logiciel graphique:

- celle qui est spécifique à l'application: le logiciel de mise en forme, qui variera donc d'un utilisateur à un autre ;
- celle qui s'occupe de l'aspect matériel: le logiciel de base, qui peut être utilisé de la même facon par des utilisateurs travaillant dans des voies totalement différentes.

Par exemple, l'écriture d'un programme de gestion ou d'un programme de conception assistée par ordinateur de nouvelles surfaces peut se faire en utilisant le même logiciel de base: le programme de gestion fournira des résultats structurés propres à son application. La visualisation de ces résultats se fait par l'appel de fonctions graphiques appartenant au logiciel graphique de base. Ces fonctions permettent de tracer des traits, colorier des surfaces... De même, la réalisation de programmes de conception assistée par ordinateur implique la construction de résultats structurés, puis l'appel des mêmes fonctions de base.

Cela peut se résumer par la figure suivante:



I.1.2 SERVICES ATTENDUS D'UN LOGICIEL GRAPHIQUE DE BASE

Un logiciel de base bien construit doit pouvoir avoir une certaine indépendance. Les exigences dans ce domaine sont d'autant plus fortes que cette indépendance ne doit pas seulement exister vis à vis de l'application mais vis à vis de tout le contexte d'utilisation, c'est-à-dire:

- l'application elle-même ;

- le matériel

  * le logiciel ne doit pas être dépendant du calculateur avec lequel on travaille. Par exemple, la modification de la vitesse de transmission entre le calculateur et l'écran graphique ne doit pas avoir d'effet sur le bon fonctionnement de ce logiciel ;

  * le logiciel doit pouvoir s'adapter aux différentes technologies des différentes consoles.

- le langage hôte

  * l'utilisateur doit pouvoir considérer ce logiciel comme une bibliothèque de sous-programmes auxquels on peut accéder quel que soit le langage utilisé dans le logiciel de mise en forme.

D'autre part, le logiciel graphique de base devra être "agréable" à utiliser, c'est-à-dire que l'utilisateur pourra se servir des primitives du logiciel sans se préoccuper de ce qu'implique chacune d'elles au niveau technique.

Enfin, le logiciel de base interactif devra être efficace pour que la notion d'interaction soit respectée dans des délais acceptables.

I.2 ETUDE DE QUELQUES LOGICIELS DE BASE

Lorsque l'utilisateur novice doit écrire une application graphique, la première chose qu'il doit faire est le choix d'un logiciel graphique de base. Ces dix dernières années, il y a un fort développement de ces logiciels (Tek82), (HP82), (LLM76), (GSP79)... Cependant, l'utilisateur retrouvera, en les comparant, de nombreuses notions communes. Nous présenterons ici quatre de ces logiciels qui, à notre avis, représentent bien la tendance actuelle. Après avoir défini certains critères de classification des primitives de ces logiciels, nous ferons ressortir les particularités de chacun d'eux.

I.2.1 CLASSIFICATION DES PRIMITIVES

Chaque primitive des logiciels graphiques de base qui nous intéressent se retrouve dans l'une de ces catégories.

a) Primitives de construction

Ces primitives permettent d'une part de construire le dessin et d'autre part d'attribuer à certaines parties de ce dessin certaines caractéristiques (ex.: attribution d'une certaine couleur à tel objet composant le dessin).

b) Primitives de consultation

Ce sont des primitives qui permettent de connaître les attributs (ou caractéristiques) attachés à un objet. Par exemple, l'utilisateur peut savoir si un objet qu'il désigne est un cercle.

c) Primitives de saisie graphique

Elles permettent, à partir du terminal graphique, de désigner un objet, ou bien de récupérer des points entrés par un dispositif quelconque (réticule, tablette à digitaliser...).

d) Primitives de visualisation

Ces primitives permettent l'affichage (ou le non affichage) de la structure graphique (ou d'une partie de cette structure) construite par le logiciel graphique de base. On suppose que l'utilisateur dispose d'un plan illimité pour représenter son dessin. La partie du plan qu'il désire visualiser est appelée "fenêtre".

D'autre part, la visualisation peut se faire non pas sur l'écran total mais sur une partie d'écran ("clôture").

Tous les logiciels de base étudiés par la suite possèdent les primitives relatives à ces notions de fenêtre et clôture.

I.2.2 LE LOGICIEL GRIGRI

Le logiciel graphique de base GRIGRI (LLM76), (LLM78) a été développé dans le laboratoire d'Informatique et de Mathématiques Appliquées de Grenoble (IMAG). Il est actuellement commercialisé et a été à la base de nombreux concepts (notamment en ce qui concerne l'indépendance vis à vis du contexte d'utilisation).

a) Primitives de construction

Les objets sont construits grâce à des primitives telles que "Points", "Texte", qui affectent une morphologie (Mar82) à l'objet au moment de sa création.

Il est également possible de donner des attributs tels que couleur, taille de caractère... (primitive "Mode").

b) Primitives de consultation

Aucune consultation d'attributs n'est possible.

c) Primitives de saisie graphique

La désignation d'un objet sur l'écran ("identifier") et l'acquisition de coordonnées ("position") sont possibles.

d) Primitives de visualisation

Deux primitives permettent d'afficher ou d'effacer des dessins ou des parties de dessins. ("afficher" et "effacer")

Remarques:

- Il existe une notion de structuration à deux niveaux des objets dans ce logiciel. En effet, les objets sont considérés comme des figures qui sont elles-mêmes des ensembles de sections (qui peuvent représenter des sous objets).

- Aucune opération géométrique n'est prédéfinie: l'utilisateur doit lui-même programmer ses fonctions de rotation, translation, zoom ...

- Le texte et les figures ne sont pas traités de la même façon: par exemple, il est impossible d'identifier (au sens désignation d'un objet) un texte.

I.2.3 LE LOGICIEL I.G.L.

I.G.L. (Interactive Graphic Library) (Hvi79), (Tek82) est un produit développé et commercialisé par la société TEKTRONIX.

Comme son nom l'indique, I.G.L. doit être plutôt considéré comme une bibliothèque de sous programmes destinés à aider l'utilisateur. Nous n'énumèrerons pas ici toutes les primitives d'I.G.L. qui sont au nombre de 175 (dont 84 sont des primitives de communication avec le système gestion de fichiers...) mais nous dégagerons ce qui fait l'originalité du logiciel.

a) Primitives de construction

Pour construire des dessins à l'aide d'I.G.L., la philosophie est plutôt d'imaginer que l'on tient une plume dans sa main que l'on déplace sans tracer (move) ou bien en traçant (draw). Certaines primitives permettent d'avoir du texte (avec éventuellement plusieurs polices de caractères) et de tracer directement des cercles, des polygones...

b) Primitives de consultation

Au sens I.G.L., un segment est constitué d'un ensemble d'objets visualisables (points, cercles, arcs de cercle, segments de droite...). Le mot "segment" ne doit pas être pris au sens "segment de droite". Il s'agit d'un ensemble d'objets que l'on groupe, constituant ainsi un "segment I.G.L".

Un segment peut être visible ou invisible: tous les objets du segment sont visibles ou invisibles.

Un segment peut être détectable (ou non): tous les objets du segment peuvent être désignés (ou non) à l'aide d'un dispositif interactif.

L'ensemble des segments est mémorisé:

- soit dans le terminal même. Dans ce cas, l'affichage est instantané car il n'y a pas de communication avec l'ordinateur hôte (ex.: Tektronix 4114).

- soit dans la mémoire de masse de l'ordinateur (ex.: Tektronix 4014).

Les seules consultations possibles se situent en fait au niveau des segments: L'utilisateur peut savoir si un segment est visible ou non, en cours de création ou créé, détectable ou non, mais il ne peut rien connaître sur le dessin contenu dans ce segment (sa couleur par exemple).

c) Primitives de saisie graphique

Une primitive est prévue pour récupérer les coordonnées d'un point à l'aide d'un dispositif quelconque (locate).

Pour l'identification d'un objet, l'objet doit faire partie d'un segment. De plus, ce segment doit être visible et détectable (cette notion de détectabilité de segments règle d'ailleurs un certain nombre de conflits lorsque la désignation porte sur deux objets en même temps).

d) Primitives d'affichage

Il n'y a pas de primitives pour afficher sur l'écran un dessin à un moment donné. C'est directement lorsque l'on construit la figure que le dessin s'affiche. Donc, pour retarder l'apparition d'une figure, il faut que l'utilisateur gère lui-même les dessins qu'il veut retarder en les mettant dans des segments invisibles.

Remarques:

- Une notion importante qui n'existait pas dans GRIGRI est introduite dans ce logiciel: celle de segmentation. Ici, les objets mémorisés sont rangés dans des segments. Les sous-segments existent également et partagent le dessin en parties de dessin. Nous avons donc, comme dans GRIGRI, la notion de structuration de la scène graphique à deux niveaux. Cependant, cette notion de segmentation est dynamique (liée à l'exécution) puisque c'est l'utilisateur qui décide du moment où il crée ses segments. Cela peut mener à une structuration des données en fonction du programme écrit alors qu'habituellement, c'est la structure du programme qui est bâtie sur les données dont on dispose. Par contre, cette notion de segmentation résoud très bien le problème d'effacement sélectif. (La mise à jour de la liste d'objets visibles est immédiate).

- Avec ce logiciel, il y a également indépendance par rapport au matériel (du moins si ce matériel est compatible avec TEKTRONIX). Cette indépendance est assurée par l'écriture de dispositifs de commandes ("drivers") différents suivant le type de matériel que l'on utilise. Ainsi, l'utilisateur

devra toujours écrire "move" lorsqu'il veut déplacer sa plume mais cet ordre sera interprété différemment suivant qu'il travaille sur un écran TEK 4114 et son clavier ou sur une table à digitaliser TEK 4662. Une primitive obligatoire dans chaque programme utilisateur indique le type de matériel sous lequel on travaille (GRSTRT) et permet au système graphique de savoir quel "driver" il utilisera.

- De nombreuses facilités sont mises à la disposition de l'utilisateur: les opérations de transformation géométriques existent même sur le texte. L'utilisateur peut afficher directement des polygones, des arcs de cercle... tout cela est dû au grand nombre de primitives offertes par cette bibliothèque.

I.2.4 G.K.S.

G.K.S. (Graphical Kernel System) (EEK79), (ISO82) est le résultat de nombreux travaux effectués en vue de normaliser les logiciels graphiques de base. Longtemps en concurrence avec la proposition de norme américaine (GSP79), c'est pour une fois la norme européenne qui a été acceptée.

a) Primitives de construction

Comme pour GRIGRI, les objets sont construits grâce à certaines primitives (polyline, texte...) qui leur affectent en même temps une morphologie (ligne brisée, texte, polygone...). Certaines primitives permettent d'affecter des attributs (remplissage,...).

b) Primitives de consultation

La notion de structuration étant assurée de la même manière que dans I.G.L., les primitives de consultation sont identiques.

c) Primitives de saisie graphique

Là encore, ces primitives sont similaires à I.G.L. La récupération des coordonnées d'un point entré interactivement ("locator") et l'identification d'un objet qui appartient à un segment ou sous-segment visible et détectable ("pick") sont possibles. Cependant, il existe une différence importante avec I.G.L. car il est possible de spécifier le mode de saisie: saisie en continu, avec attente, avec visualisation d'écho...

d) Primitives d'affichage

Contrairement aux logiciels précédents, l'utilisateur peut choisir son mode de visualisation: on peut afficher comme dans I.G.L. dès que le dessin est construit, attendre la prochaine interaction ou bien reporter sans cesse l'affichage et le demander explicitement avec la primitive "redraw" (similaire à GRIGRI).

Remarques:

- Les mêmes remarques que pour I.G.L. en ce qui concerne la structuration à l'aide de segments restent valables.

- Le point fort de ce logiciel et sa nouveauté par rapport aux autres logiciels est qu'il permet des modes différents autant pour saisir les données que pour afficher les résultats.

- Le texte est traité de la même manière que les dessins.

I.2.5 LE LOGICIEL CLOVIS

Contrairement aux logiciels évoqués précédemment, CLOVIS n'est pas encore commercialisé. Créé à l'IMAG en 1982 (Mar82), ce logiciel est encore à l'état de prototype. Ce que nous exposons ici reflète son état à l'heure actuelle mais nous sommes conscients que certains aspects seront rapidement reconsidérés. La liste complète des primitives réalisées jusqu'alors peut être trouvée dans (Gra83).

La philosophie de CLOVIS étant différente de celle des autres logiciels, nous allons rapidement en exposer les concepts principaux:

L'utilisateur devra toujours considérer qu'il compose un dessin structuré. N'importe quel dessin peut en effet être décomposé en arbre. (arbre à un seul niveau si le dessin n'a aucune structure). Pour composer un dessin, l'utilisateur devra tout d'abord décrire la structure de son dessin grâce à certaines primitives particulières, puis affecter à certains noeuds de cette structure des attributs. C'est seulement à ce moment-là que l'on saura si l'objet est composé de lignes brisées, cercles, texte... Ainsi, on remarque que ce n'est pas le fait de construire un objet qui lui donne sa forme. L'utilisateur peut retarder cette attribution de forme autant qu'il le désire. Les attributs morphologiques ne sont pas les seuls que l'on peut affecter aux noeuds de l'arbre. En effet, la couleur, le type de tracé, la taille des caractères... sont également attachés aux éléments de la structure. Toutes les opérations primitives de CLOVIS se font sur des arbres ou des sous-arbres, ce qui implique que l'utilisateur a, à sa disposition, un certain nombre de fonctions qui lui permettent de "naviguer" dans son arbre. Il pourra ainsi ne visualiser qu'une partie de son dessin, se placer dans un certain contexte où les seuls objets identifiables sont ceux qui appartiennent au sous-arbre pointé par le contexte considéré, etc.

Classifions à présent les primitives de CLOVIS.

a) Primitives de construction

Elles sont de deux types: les primitives de construction ou de modification de la structure ("structures", "détruire") et les primitives d'attribution de morphologie ou d'aspect ("attribut"...) à un noeud ou à un ensemble de noeuds.

b) Primitives de consultation

Il n'est pas actuellement possible de consulter les attributs attachés à un noeud mais il semble que cela sera réalisé dans un futur proche.

c) Primitives de saisie graphique

Une primitive permet d'acquérir les coordonnées d'un point entré graphiquement ("coordonnées") par un dispositif physique quelconque.

Une autre primitive permet d'identifier un objet du contexte dans lequel on se trouve. Elle rend comme résultat le chemin qu'il faut parcourir dans l'arbre à partir de la racine de ce contexte.

d) Primitives d'affichage

La primitive "afficher" permet de visualiser toute la partie de la structure qui est donnée en paramètre. Si aucune morphologie n'est attachée à un noeud, rien ne s'affiche pour ce noeud.

Remarques:

L'avantage principal de ce logiciel graphique de base est sa notion de forte structuration. Cela permet notamment, lorsque l'on déplace (translation, rotation, zoom) un sous-arbre, de déplacer tout un ensemble d'objets formant un tout logiquement (si la structure de l'utilisateur est correctement construite, bien sûr).

Il y a de plus une forte cohérence entre les primitives puisqu'elles portent sur le même genre de paramètres (des sous-arbres).

Nous pouvons cependant regretter l'absence de possibilité de consultation des paramètres qui semble n'être que momentanée.

Une autre contrainte se situe au niveau du poste de travail car le logiciel ne permet actuellement pas de faire des entrées graphiques sur plusieurs matériels physiques (réticule + tablette + ...) mais ne permet qu'un seul mode de saisie (grâce au réticule). Là aussi, nous pensons que cette lacune découle de la nouveauté du logiciel et qu'elle sera rapidement comblée.

Enfin, l'aspect communication n'est pas encore soigné: les notions de fenêtre et de clôture ne sont pas encore prises en compte, ce qui empêche de travailler avec plusieurs zones sur l'écran. La réalisation d'une zone de communications paraît indispensable puisqu'elle éviterait de faire apparaître du texte n'appartenant pas au dessin dans la zone de visualisation.

I.3 CHOIX D'UN LOGICIEL GRAPHIQUE DE BASE POUR L'EDITEUR GRAPHIQUE DE CASCADE.

L'application graphique à réaliser pour le projet CASCADE a nécessité le choix d'un logiciel graphique de base. Les problèmes principaux auxquels nous avons été confrontés sont

- la disponibilité du logiciel ;
- le prix du logiciel ;
- l'aptitude du logiciel à pouvoir faire ce que l'on attend pour notre application ;
- la fiabilité du logiciel.

Nous avons eu à choisir entre les différents logiciels décrits ci-dessus (ou dérivés). Pour des raisons techniques et financières, nous avons écarté un certain nombre d'entre eux.

Le logiciel CLOVIS nous a semblé le plus approprié à notre application: nous avons été séduits par la possibilité qu'il offrait de structurer les dessins en arbres. En effet, la structure de notre application était la même et pouvait servir de base à la structure graphique. Le problème était surtout la fiabilité du logiciel. Cependant, le fait qu'il était développé dans l'équipe était important car en cas d'erreurs, la maintenance serait assurée rapidement. Attirés par la nouveauté et l'originalité des concepts de CLOVIS, nous l'avons donc adopté pour réaliser notre éditeur graphique.

CHAPITRE II

L'EDITEUR GRAPHIQUE DE CASCADE AU NIVEAU STRUCTUREL

II.1 But de l'éditeur

II.2 Présentation des concepts de CASCADE et de leur équivalence graphique

- II.2.1 Modularité
- II.2.2 Interconnexions entre modules
- II.2.3 Attributs
- II.2.4 Porteuses
- II.2.5 Compléments de langage

II.3 Présentation d'un scénario pour l'entrée graphique

- II.3.1 Mode de classification des commandes d'un éditeur
- II.3.2 Organisation des commandes de l'éditeur en menus
- II.3.3 Exemple d'utilisation d'EDICAS
- II.3.4 Classification des commandes d'EDICAS

II.4 Traduction en langage CASCADE

- II.4.1 Structure graphique attachée à une description
- II.4.2 Traduction à partir de la structure interne d'une description
- II.4.3 Exemple de traduction

II.5 Passage du texte CASCADE à une représentation graphique correspondante

II.6 Réalisation. Résultats et évaluations.

CHAPITRE II

L'EDITEUR GRAPHIQUE DE CASCADE AU NIVEAU STRUCTUREL

II.1 BUT DE L'EDITEUR

En 1982, alors que le noyau du langage CASCADE est complètement défini (Equ82), il s'avère utile de définir un moyen graphique pour représenter d'une manière plus visuelle l'information que l'on peut définir avec le langage textuel CASCADE. Comme le concepteur est habitué à raisonner avec des schémas, il paraît important de lui permettre d'entrer le dessin de l'objet qu'il désire représenter à la place d'une description textuelle équivalente. C'est pour cette raison qu'un éditeur graphique pour CASCADE a été spécifié (Mar83a) et réalisé (Mar84a), (Mar84b).

Toute description d'objet peut être partagée en deux parties:

- la "description structurelle" qui donne une vision de la structure de l'objet;
- la "description comportementale" qui permet de décrire le fonctionnement de l'objet et les contraintes à respecter.

Suivant le niveau auquel on désire décrire son objet (logique, électrique...), les deux parties de la description sont plus ou moins importantes. Ainsi, au niveau "portes logiques" du langage CASCADE (BMV83), la partie description comportementale est vide puisqu'il s'agit de décrire un ensemble de portes logiques connectées entre elles (structurel seulement).

La partie structurelle est assez simple à décrire graphiquement contrairement à la partie comportementale où les études et la bibliographie sont moins nombreuses (Que81), (Sif79). De plus, ces représentations graphiques du comportement existent principalement pour représenter le parallélisme, problème que nous ne considérons pas ici.

Nous proposons, dans ce chapitre, de présenter ce qui, dans cet éditeur, permet d'obtenir la partie structurelle. Pour cela, nous nous baserons sur les spécifications d'un éditeur permettant d'entrer graphiquement des objets au niveau "portes logiques". La représentation graphique de la partie comportementale sera exposée dans le chapitre III.

II.2 PRESENTATION DES CONCEPTS DE CASCADE ET DE LEUR EQUIVALENCE GRAPHIQUE

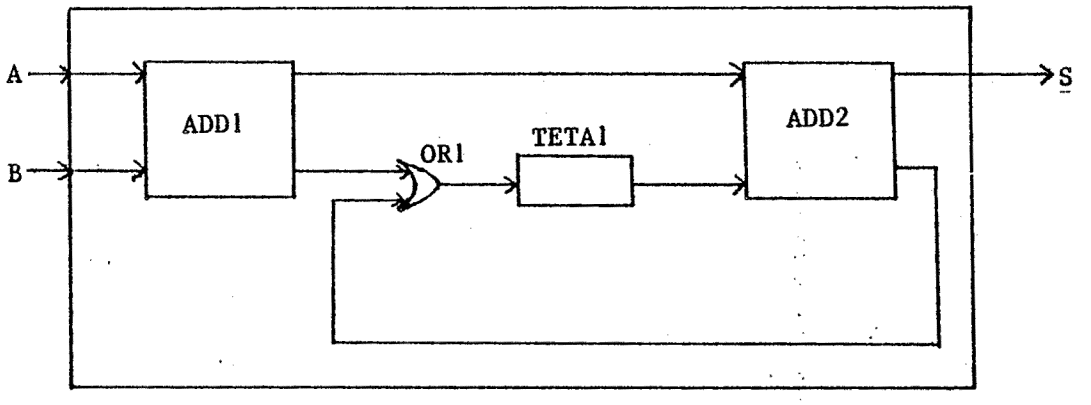
Nous introduisons ici les principaux concepts du langage CASCADE: les notions de modules, de connexions entre modules, d'attributs, de porteuses, et de compléments de langage sont décrites succintement. Le détail est exposé dans (Bor81) et (Mer83). Nous associons à chacune de ces notions une représentation équivalente.

II.2.1 MODULARITE

Dans la philosophie de CASCADE, un objet forme un tout. Il doit avoir une signification propre et être simulable indépendamment.

La structure des objets étant modulaire dans le langage CASCADE, nous pouvons représenter tout objet par un ensemble de sous-objets qui peuvent être à leur tour décomposés. Nous obtenons ainsi un arbre appelé "arbre des imbrications".

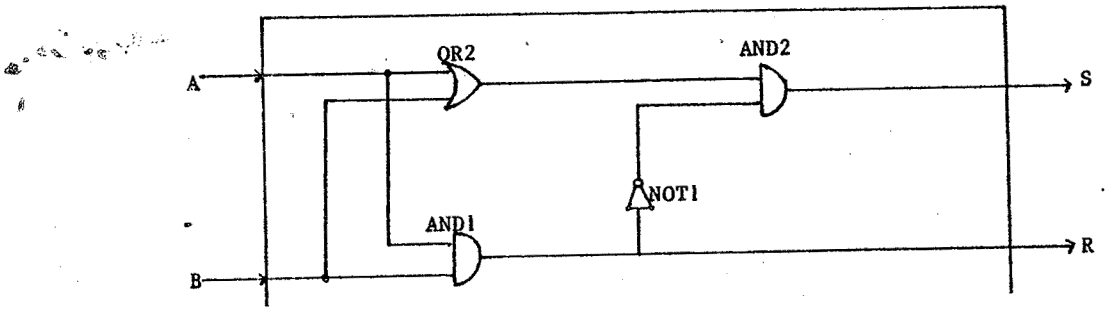
Considérons l'exemple d'un "additionneur série" (Mei71). Il peut être représenté par:



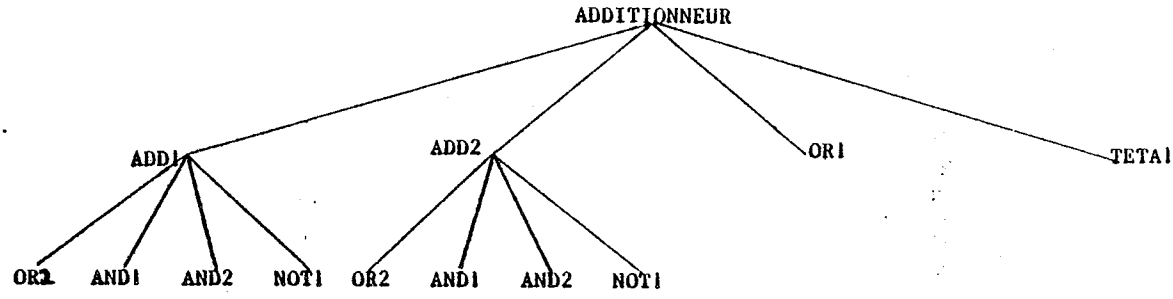
Les bits successifs des deux nombres à additionner arrivent par paires espacées de teta unités de temps, poids faible d'abord.

Les modules composant l'additionneur série sont: ADD1, ADD2, OR1, TETA1.

Certains éléments sont à nouveau décomposables. Ainsi, ADD1 et ADD2 sont chargés de faire l'addition entre deux bits et rendent un résultat et une retenue. Ils peuvent être décomposés de la manière suivante:



Si l'on considère que la décomposition s'arrête à ce niveau, c'est-à-dire que toutes les boîtes feuilles de l'arbre des imbrications sont primitives (ne peuvent plus être décomposées au niveau de description qui nous intéresse), nous obtenons l'arbre des imbrications suivant:



La racine de l'arbre représente le modèle global. Ce modèle est appelé description.

Chaque boîte associée à un noeud de l'arbre est un exemplaire d'une description définie par ailleurs ou primitive pour le niveau de langage considéré.

Par exemple, ADD1 et ADD2 sont deux exemplaires d'une même description ADD.

Dans le langage CASCADE (Equ82), ceci s'exprime par

```
description ADDITIONNEUR...
    corps
        externe ADD, OR, TETA;
        unites ADD ADD1, ADD2;
               OR OR1;
               TETA TETA1;
            .
            .  <suite du corps>
            .
    fin

description ADD...
    corps
        externe OR, AND, NOT
        unites OR OR2;
               AND AND1, AND2;
               NOT NOT1;
            .
            .
            .
    fin
```

On précise donc toutes les descriptions externes qui sont utilisées dans la description, puis on déclare chaque boîte de l'arbre en précisant de quelle description externe elle est un exemplaire.

Il est possible de regrouper des exemplaires de boîtes sous forme de tableaux si elles sont issues d'un même modèle et si elles jouent le même rôle au niveau structurel.

Exemple d'utilisation de boîtes dimensionnées:

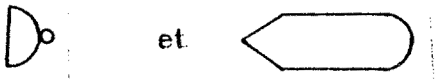
unités OR OREX[1:3]

déclare un ensemble de trois exemplaires de description OR appelés OREX[1], OREX[2] et OREX[3].

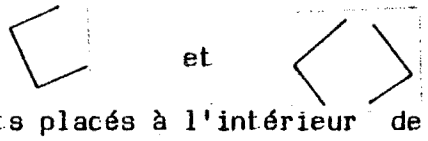
Représentation graphique associée à la modularité

A chaque description est associée une forme (ou un contour) qui permet visuellement d'identifier un objet. Ce contour est obligatoirement fermé. Il est constitué d'une suite de lignes courbes ou brisées qui sont non disjointes.

Exemples



sont des contours corrects

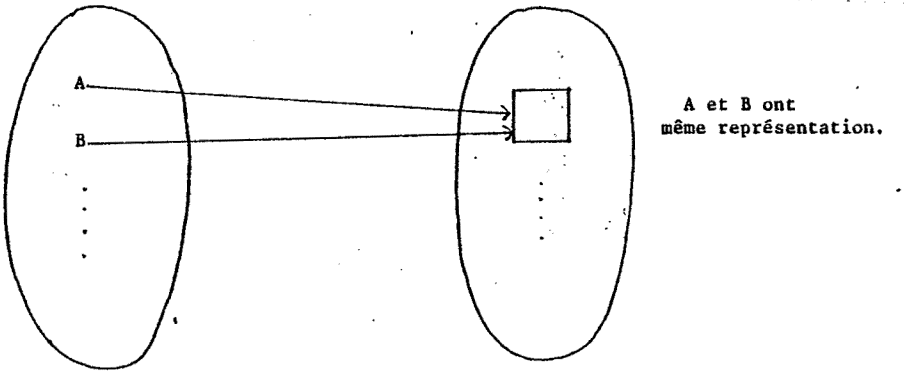


alors que et ne le sont pas.

Les objets placés à l'intérieur de la zone délimitée par le contour appartiennent à la description.

Deux descriptions différentes peuvent avoir le même contour associé. Cela prouve qu'il n'y a pas une bijection entre l'ensemble des descriptions et l'ensemble des contours.

Exemple



Cela implique que même dans la représentation graphique, il faut garder le nom de la description représentée (le contour ne suffit pas à l'identification d'un objet).

L'objet englobant étant formé d'un ensemble de sous-objets, tous les sous-objets seront inclus dans la zone délimitée par le contour englobant. Il y a ici une contrainte très forte de topologie pour

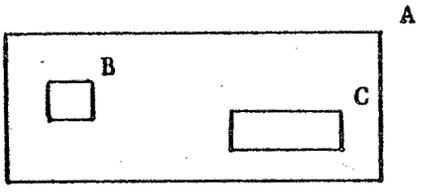
exprimer la notion de décomposition. Le seul élément qui se trouve au coin supérieur droit à l'extérieur de la boîte est le nom de la boîte pour des raisons de lisibilité.

Exemple

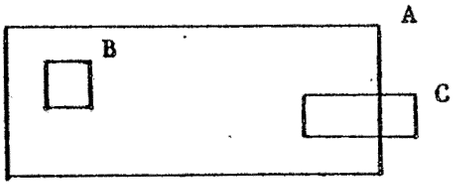
Soit l'objet A qui peut être décomposé avec les objets B et C.

exemple de

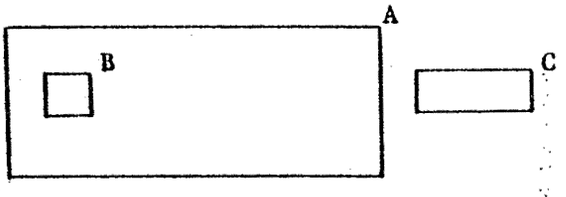
- représentation correcte



- représentation sans signification



- représentation n'exprimant pas la même structuration



Remarques:

Tous les exemplaires d'une description ont la même représentation graphique que cette description. Les seules opérations possibles sur cette représentation graphique sont:

- le facteur d'échelle pour pouvoir placer des exemplaires avec la taille désirée qui est souvent fonction du nombre de sous-objets de la description;
- la rotation et la translation qui permettent d'obtenir des dessins moins confus.

Evidemment, ces opérations devront respecter la contrainte d'appartenance des objets exprimée ci-dessus. On ne pourra pas par exemple, translater une boîte en dehors de la description englobante ou grossir un exemplaire de telle sorte qu'il "déborde" de la boîte englobante.

II.2.2 INTERCONNEXIONS ENTRE MODULES

Lorsque des exemplaires de descriptions sont utilisés dans une description englobante, il faut qu'ils puissent communiquer entre eux. Cette communication ne peut pas se faire à l'aide de variables globales puisque les objets doivent former un tout.

L'"interface" est le seul moyen de communication entre les différents composants de la description englobante. Elle est constituée de "pattes" d'entrée, de sortie, bidirectionnelles ou non directionnelles. Ces pattes peuvent être regroupées entre elles si elles sont du même type (les informations qui circulent sont du même type et les pattes sont de même sens). On obtient ainsi des tableaux d'éléments d'interface éventuellement à plusieurs dimensions. Par exemple, "in VARLOG E[1:15]" déclare quinze éléments d'interface E[1]...E[15] qui sont des variables logiques (VARLOG) en entrée (in).

Une vision possible de cette notion est de considérer la construction d'une description englobante comme une juxtaposition de boîtes noires (les exemplaires de descriptions externes) dont chacune a des "pattes" de communication avec les autres. Seules les listes et caractéristiques de ces "pattes" (interfaces) sont connues pour chaque boîte.

Pour chaque description, la liste de ses éléments d'interface constitue l'"interface formelle". L'"interface effective" d'un exemplaire de description est définie lorsque l'on fixe les différentes connexions entre les modules. Ces notions sont détaillées dans (Bor81).

On peut faire communiquer les exemplaires de descriptions grâce à leur interface de deux manières différentes:

- établir des liaisons au moment de la déclaration des exemplaires de description en créant des liaisons permanentes (il y a synonymie entre les éléments d'interface reliés entre eux) pour des descriptions toutes écrites au même niveau de langage.

- établir des liaisons dans la partie "relations" (chapitre III) qui peuvent être conditionnées et qui ne sont pas permanentes.

Exemple: l'additionneur série.

Au niveau le plus englobant (description ADDITIONNEUR), l'interface est composée de deux entrées (A et B) et d'une sortie (S). Cela sera décrit par:

```
description ADDITIONNEUR (in VARLOG A, B;  out VARLOG S)
corps
   .
   .
   .
fin
```

VARLOG indique que A, B, et S sont du type "variable logique" (cf II.2.4.).

Ainsi, ADDITIONNEUR peut être utilisé, à son tour, comme description externe d'un module encore plus englobant (multiplication, par exemple).

ADD est une description qui permet d'additionner deux bits et de sortir le résultat et la retenue. ADD a une interface formelle conposée de deux entrées et de deux sorties:

```
description ADD (in VARLOG A, B;  out VARLOG S, R)
corps
   .
   .
   .
fin
```

On a également:

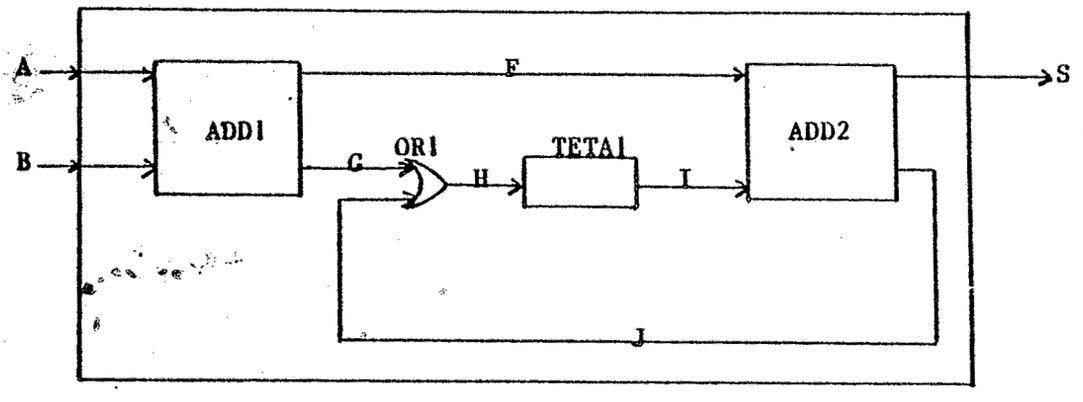
```
description OR (in VARLOG E1, E2;  out VARLOG S);
description TETA (in VARLOG E;  out VARLOG S);
```

Pour connecter un élément d'interface d'une boîte avec un élément d'interface d'une autre boîte, de manière permanente, deux solutions sont envisageables:

- connecter chacun des éléments d'interface à une même variable locale;
- connecter directement les éléments entre eux (utilisation de la notation pointée: l'extrémité de la connexion est indiquée par le nom de la boîte postfixé par le nom de l'élément d'interface).

Exemple:

- en utilisant des variables locales:



Les connexions se décrivent de la manière suivante:

```
description ADDITIONNEUR (in VARLOG A, B;  out VARLOG S)
corps
   externe ADD, OR, TETA;
   decl VARLOG F, G, H, I, J;
   unites ADD ADD1(A, B, F, G), ADD2(F, I, S, J);
          OR OR1(G, J, H);
          TETA TETA1(H, I);
      .
      .
      .
fin
```

Les variables locales F, G, H, I, J permettent d'établir les connexions par synonymie entre les interfaces des différents composants. F, par exemple, permet de connecter la première sortie de ADD1 avec la première entrée de ADD2.

- en utilisant la notation pointée:

```
description ADDITIONNEUR (in VARLOG A, B;  out VARLOG S)
corps
   externe ADD, OR, TETA
   unites ADD ADD1(A, B,,),
              ADD2(ADD1.S,,S,);
          OR OR1(ADD1.R,ADD1.R);
          TETA TETA1(OR1.S,ADD2.B);
      .
      .
      .
fin
```

On remarque qu'il n'y a pas de référence avant: on ne peut pas faire référence à un objet lorsqu'il n'est pas encore placé. Ainsi la liaison entre ADD1.S et ADD2.A n'est effective que lors du placement de ADD2. Lorsqu'on place l'exemplaire ADD1, on ne marque rien dans l'interface effective pour les éléments que l'on ne connecte pas.

Représentation graphique associée à l'interconnexion entre modules

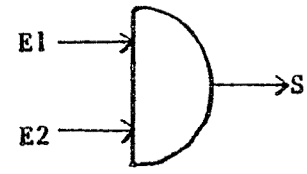
a) Les éléments d'interface

L'interface étant le seul moyen de communication entre les modules, il est intéressant de situer les éléments de l'interface sur le contour associé aux descriptions. C'est en effet par l'interface que circule toute information en provenance de l'extérieur vers le module ou du module vers l'extérieur.

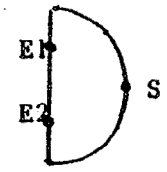
Il est possible de représenter les éléments d'interface de plusieurs manières. Nous avons choisi une représentation qui permet, en plus de la connaissance de l'emplacement de l'élément d'interface, la connaissance de son sens (entrée, sortie...). Cette représentation consiste à placer une flèche dont l'origine ou l'extrémité (sortie ou entrée) est située sur le contour. Si l'élément d'interface est non directionnel, il n'y aura pas de flèche

mais un trait (-). S'il est bidirectionnel, il sera représenté par une double flèche (<->).

Ainsi, pour la représentation de la porte ET à deux entrées, nous aurons:

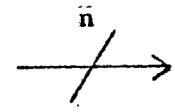


Une autre représentation possible aurait été:



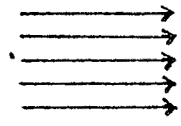
mais il y a moins de sémantique attachée au dessin dans ce cas-là.

La représentation d'un élément d'interface dimensionné ne peut pas se faire à l'aide d'une seule flèche puisque plusieurs éléments sont regroupés entre eux. Deux représentations sont possibles:

- une représentation compactée 

  où n est le nombre de composants de l'élément d'interface;

- une représentation développée



  où il y a autant de flèches que de composants dans l'élément d'interface.

Remarques:

- la représentation d'un élément d'interface nécessite le stockage du nom de l'élément d'interface dans la structure;
- la signification du graphisme de l'élément d'interface ne dépend pas seulement du dessin de cet élément: en effet, dans l'exemple, nous voyons que "----->" peut signifier qu'il s'agit d'un élément d'interface d'entrée ou de sortie. Il faut également considérer le placement de ce dessin par rapport au contour de la boîte.
- nous verrons dans le chapitre III que la sémantique associée au graphisme peut être encore plus grande puisque l'on peut établir une correspondance entre les couleurs et les types pouvant circuler par les éléments d'interface.

b)Les connexions

Pour connecter deux éléments, il suffit graphiquement de les relier par un ensemble de segments de droites ayant pour origine un des deux éléments et pour extrémité l'autre. Deux contraintes topologiques doivent cependant être respectées pour des raisons technologiques:

- les segments de droite sont horizontaux ou verticaux;
- les segments de droite ne traversent pas les modules déjà placés.

De nombreux travaux ont été effectués pour qu'en respectant ces contraintes, il soit possible de tracer de manière automatique le "meilleur chemin", d'après certains critères (nombre de croisements, longueur...) entre deux éléments d'interface (Lee61), (Ser82). Le choix de notre algorithme de routage et son implémentation est expliqué dans (Gui84).

Une notion graphique associée à l'interconnexion des modules doit être précisée ici. Bien que cela ne soit pas obligatoire pour l'utilisateur, il arrive souvent que les modules composant un modèle soient regroupés topologiquement suivant le critère de connectivité entre modules: deux modules seront très proches l'un de l'autre s'il existe de nombreuses connexions entre eux. Des algorithmes reposant sur ce critère permettent de faire du placement automatique de modules (Ser82). Nous n'avons pas choisi de mettre en oeuvre de tels algorithmes en laissant le concepteur libre de son placement. En fait, lorsqu'il compose son dessin, il regroupe effectivement ses boîtes de manière à obtenir le plus de clarté possible. Il est également guidé par la connaissance de la structure de l'objet.

II.2.3 ATTRIBUTS

Dans le langage CASCADE, on a la possibilité de décrire une famille de modèles à la place d'un modèle unique en associant des attributs à la description d'un modèle.

La famille de modèles décrite n'est pas quelconque puisque tous les modèles auront même structure et même comportement. Les seules différences se situeront au niveau de leur dimension ou de leur durée.

Exemple 1: attribut ayant une influence sur la durée

Reprenons la boîte TETA1 exemplaire du modèle TETA.

Ce modèle retarde l'information disponible en entrée pendant un certain temps avant de la rendre en sortie. Si nous considérons que ce temps est un attribut du modèle, nous obtenons ainsi une famille de modèles tous destinés à retarder l'information mais d'un temps différent.

```
description TETA (ENT DELAI) (in VARLOG E; out VARLOG S);
corps
   S .= % DELAI ! E
fin
```

L'opérateur % est l'opérateur de retard. Ici, il retarde E de DELAI unités de temps avant de le rendre disponible sur S.

Ainsi, dans l'additionneur, lors de la déclaration d'un exemplaire de TETA, il faut aussi fixer l'attribut:

```
unites .
       .
       .

      TETA (15) TETA1 (H, I);
       .
       .
       .

fin
```

permettra de décrire un additionneur série où les bits des nombres à ajouter arrivent espacés de 15 unités de temps.

Remarquons que l'additionneur lui-même peut avoir un attribut:

```
description ADDTIONNEUR (ENT TEMPSECART) (...)
    .
    .
    .

  unites .
         .
         .

        TETA (TEMPSECART) TETA1 (H, I);
         .
         .
         .

  fin
```

définit une famille de modèles d'additionneurs série différant entre eux par le temps où deux données successives sont disponibles.

Exemple 2: attribut ayant une influence sur les éléments d'interface

Considérons la description suivante:

```
description ANDNENTR (ENT N)
                     (in VARLOG E(1:N);  out VARLOG S);
   .
   .
   .
```

Cette description permet de décrire un "AND" à N entrées. Nous voyons que les N entrées sont de même type et sont regroupées sous le nom de E.

Lors de l'utilisation d'un exemplaire de cette description, il faudra fixer N:

```
description UTILISATION... declare VARLOG A[1:3], B[1:2], C, D, E;
corps
   externe ANDNENTR,...
   unites ANDNENTR (3) AND1 (A, C)
          ANDNENTR (2) AND2 (B, D)
          ANDNENTR (2) AND3 (B, E)
      .
      .
      .
fin
```

Dans cette description, on utilise trois exemplaires de la description ANDNENTR, dont deux sont des AND à deux entrées (AND2 et AND3) et un AND à trois entrées (AND1).

Représentation graphique des attributs

Une description paramétrée par un attribut définit une famille de descriptions. La représentation graphique sera la même pour toute description de la famille en ce qui concerne la forme (ou le contour) associée à la description. Ceci respecte le fait que toutes les descriptions d'une même famille jouent le même rôle structurellement.

Si l'un des attributs de la description se rapporte à un élément d'interface ou à une boîte imbriquée, alors il y a une traduction graphique de cet attribut.

a) Cas où l'attribut porte sur un élément d'interface

Un élément d'interface est dimensionné et l'on ne connaît pas le nombre de sous-objets composant cet élément puisque ce nombre n'est fixé qu'au moment de l'utilisation d'un exemplaire du modèle.

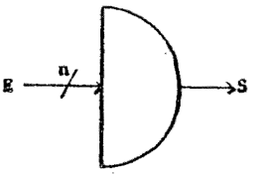
Les deux représentations graphiques utilisées pour les éléments dimensionnés (paragraphe II.2.2) deviennent alors:

- n
  ---/---> où n est le nombre de composants de l'élément d'interface, pour la représentation compactée;

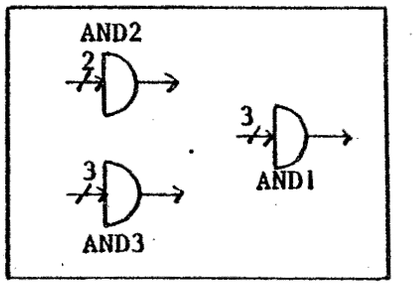
- puisque le nombre n est inconnu, des points sont placés entre le premier et le dernier élément composant l'élément d'interface, pour la représentation développée.

Exemple: "AND" à n entrées

- 1ère représentation:

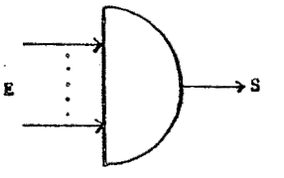


représentation de ANDNENTR

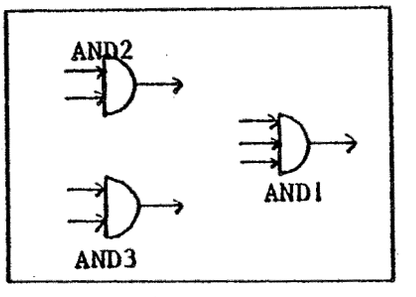


représentation de UTILISATION

- 2ème représentation



représentation de ANDNENTR



représentation de UTILISATION

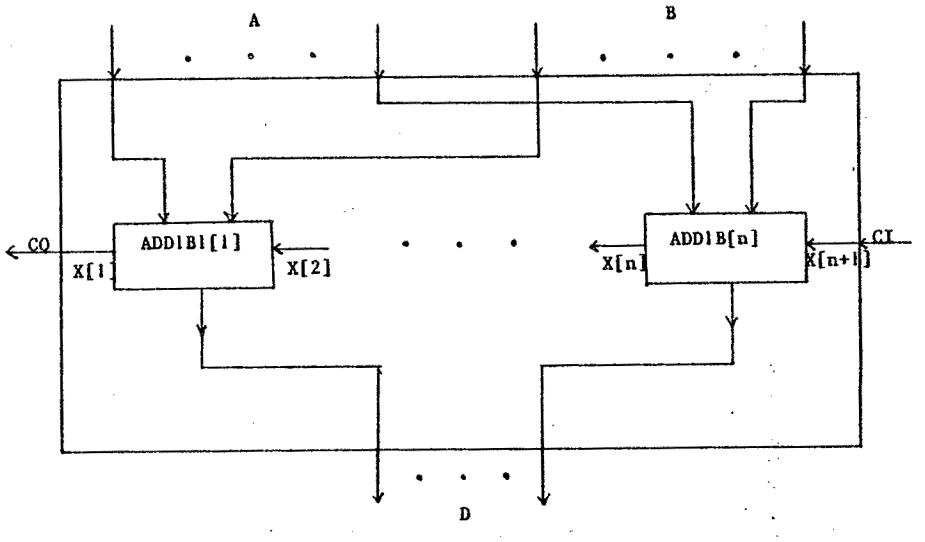
b) Cas où l'attribut porte sur une boîte imbriquée

La représentation graphique d'une boîte imbriquée dimensionnée est connue en suivant le même principe que pour la représentation graphique des éléments d'interface dimensionnés.

La boîte dimensionnée est éclatée en N boîtes en intercalant des pointillés si N n'est pas connu.

Exemple: additionneur parallèle de n bits

On cherche à construire une famille d'additionneurs paramétrés par n, nombre de bits reçus en entrée pour chaque opérande et nombre de bits en sortie, en utilisant des cellules d'additionneur un bit (n cellules). La représentation graphique est la suivante:



Remarque: La représentation compactée n'est pas possible ici: en représentant N boîtes avec une seule, on perd toute l'information concernant les connexions existant entre les N boîtes.

II.2.4 PORTEUSES

La notion de porteuse est une généralisation de la notion de variable (Bor81). Une porteuse est un objet faisant partie du modèle (registre, fil électrique, variable...), qui contient une valeur (logique, entière, réelle...) et qui est caractérisée par les opérations permettant une lecture ou une modification de cette valeur. Un couple (t, d) est associé à chaque type de porteuse. t est le type de la valeur contenue dans la porteuse et d est la valeur par défaut de la porteuse.

Exemples:

1) VARLOG = VARIABLE (LOGIC4, `x)

Le type de porteuse VARLOG déjà rencontré dans les exemples précédents permet de définir des variables ne pouvant contenir que des informations de type LOGIC4 (= {`0, `1, `x, `z }) et qui sont initialisées avec `x, c'est-à-dire la valeur indéfinie.

2) FIL = FILELEC (REEL, 0)

Ce type de porteuse permet de définir un fil électrique portant des intensités réelles et dont la valeur par défaut est 0.

Une liste complète des types primitifs de porteuses actuellement pris en compte dans CASCADE est donnée dans (Bre83).

Représentation graphique des porteuses

Une porteuse étant un objet indépendant, une représentation graphique lui est associée. Cette représentation graphique est connue si le type de porteuse est prédéfini. Elle est donnée par l'utilisateur dans le cas où il définit lui-même un nouveau type de porteuse.

Exemple: au niveau structurel, toutes les lignes transportant des signaux sont représentées par des lignes brisées (voir connexions entre modules, paragraphe II.2.2)

II.2.5 COMPLEMENT DE LANGAGE

Pour modéliser un système, il est nécessaire de disposer de plusieurs niveaux d'abstraction. En effet, pour modéliser un système au niveau logique et au niveau électrique, les objets à décrire sont différents et l'on n'aborde pas le problème de la même façon. Une certaine classification de ces différents niveaux d'abstraction a déjà été proposée (Mer73).

Ces différents niveaux d'abstraction sont représentés dans le langage CASCADE par des compléments de langage système qui rendent disponibles certaines instructions et indisponibles d'autres, qui prédéfinissent un ensemble de types, de fonctions et de descriptions. Par exemple, au niveau "portes logiques", les portes AND, OR... sont prédéfinies. Le type de porteuses VARLOG vu précédemment l'est aussi.

L'utilisateur peut cependant avoir besoin de certaines descriptions, de certains types ou de certaines fonctions qui ne sont pas prédéfinies mais qu'il utilise très fréquemment. Dans ce but, la notion de complément de langage peut être étendue en donnant à l'utilisateur le moyen de se définir lui-même ses propres compléments de langage. Il définira ainsi des objets qui, pour lui, seront prédéfinis lorsqu'il décrira une description dans son complément de langage.

Exemple:

```
lanref POLO
langage PORTE 1
    .
    .
    . definitions
```

Ceci définit le complément de langage PORTE1.

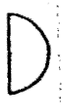
```
lanref POLO PORTE1
    .
    .
    . descriptions
```

permet de décrire des descriptions utilisant les objets prédéfinis dans PORTE1.

Représentation graphique du complément de langage

La représentation graphique d'un complément de langage est composée de toutes les représentations graphiques associées aux éléments appartenant au complément de langage.

Si le complément de langage est un complément de langage "système" (il est déjà prédéfini), il n'y a pas de problème car les représentations graphiques associées aux descriptions prédéfinies sont aussi prédéfinies dans l'éditeur. Au niveau POLO (portes logiques), par exemple, la représentation graphique de AND est



etc...

Par contre, si le complément de langage est défini par l'utilisateur, c'est l'utilisateur lui-même qui devra associer des éléments graphiques à chacune des descriptions qu'il définit.

Nous distinguons deux sortes d'utilisateurs:

A) Ceux qui sont capables de définir des compléments de langage (équipe CAO,...).

B) Ceux qui écrivent seulement des descriptions en précisant le niveau et les compléments de langage qu'ils veulent utiliser (concepteurs de circuits).

Dans notre réalisation, toutes les données sont stockées dans des fichiers.

Ces fichiers ont un rôle différent identifié par leur suffixe. Par exemple, ex.dat contiendra la structure graphique de ex; ex.cas contiendra la description textuelle associée à ex.

Pour chaque complément de langage, un fichier contient la liste des noms de tous les objets prédéfinis dans ce complément de langage.

Pour un complément de langage "utilisateur", ce fichier contient en plus le nom du complément de langage dont il "dérive". (Dans l'exemple précédent, Portel dérive de POLO, c'est-à-dire que tous les éléments prédéfinis dans POLO le sont également dans Portel.)

Tous les fichiers dans lesquels sont stockées les structures graphiques des descriptions prédéfinies (un fichier contient une description) sont protégés en écriture contre tout utilisateur dans le cas d'un complément de langage "système", et contre les utilisateurs de type 8 dans le cas d'un complément de langage utilisateur.

Exemple:

Au niveau POLO, les descriptions prédéfinies sont AND, OR, NOT, NAND, XOR, XNOR, NOR.
Supposons qu'au niveau Portel les descriptions prédéfinies soient DESC1, DESC2.
Si l'utilisateur décrit son circuit avec le complément de langage Portel, l'ensemble des boîtes prédéfinies dont il dispose est décrit dans le fichier de complément de langage Portel et dans le fichier du complément de langage dont dérive Portel (ici POLO).
L'ensemble est donc AND, OR, NOT, NAND, NOR, XOR, XNOR, DESC1, DESC2.

Après avoir vu les concepts de CASCADE et leur équivalence graphique, nous exposons dans le paragraphe suivant comment définir et classifier les commandes d'un éditeur permettant de définir des descriptions graphiques.

II.3 PRESENTATION D'UN SCENARIO POUR L'ENTREE GRAPHIQUE

Nous exposons d'abord une classification des commandes de l'éditeur graphique de CASCADE. Puis, nous définissons les actions associées à chaque commande. Nous organiserons alors les commandes en menus, ce qui donnera une vision "utilisateur" de l'éditeur. Enfin, nous donnerons un exemple d'utilisation de l'éditeur EDICAS.

II.3.1 MODE DE CLASSIFICATION DES COMMANDES D'UN EDITEUR

Les commandes des éditeurs sont en général divisées en trois groupes:

- Le premier groupe (G1) contient les commandes de mise au propre du dessin. Le cadrage, le titre du dessin, l'ajout de couleurs non significatives sont des commandes qui appartiennent à ce groupe. Toutes ces commandes n'influent que sur l'affichage et en aucun cas sur la structure du dessin affiché.

- Le deuxième groupe (G2) contient les commandes qui modifient la scène graphique. Il arrive cependant qu'en modifiant la scène graphique, on modifie également la sémantique de l'objet. Par exemple, si on donne une signification à la couleur, le fait de changer de couleur influera sur l'objet.

Le groupe G2 sera donc séparé en deux sous-groupes (G2a et G2b) qui contiendront :

* les commandes modifiant des informations graphiques sans influence sur la sémantique de l'objet (G2a). Par exemple, la translation d'une boîte englobée est sans signification sémantique;

* les commandes modifiant des informations graphiques en influant également sur la sémantique de l'objet (G2b). (exemple de la couleur).

- Le troisième groupe (G3) contiendra des commandes très puissantes qui modifieront les segments où sont rangées les structures graphiques des descriptions. Les commandes de validation d'une description, par exemple, appartiennent à ce groupe.

Nous présentons, à présent, l'ensemble des commandes utilisées dans l'éditeur.

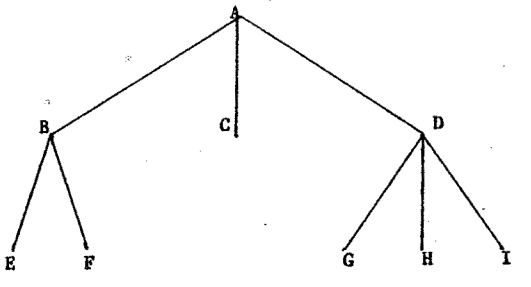
II.3.2 ORGANISATION DES COMMANDES DE L'EDITEUR EN MENUS

L'éditeur graphique de CASCADE n'est pas un éditeur où toutes les commandes sont "à plat". L'utilisateur ne peut pas, en effet, appeler n'importe quelle action à n'importe quel moment. (De nombreux éditeurs de texte ont un ensemble de commandes "à plat": Ed, Ted sous le système MULTICS, Edit sous RSX...).

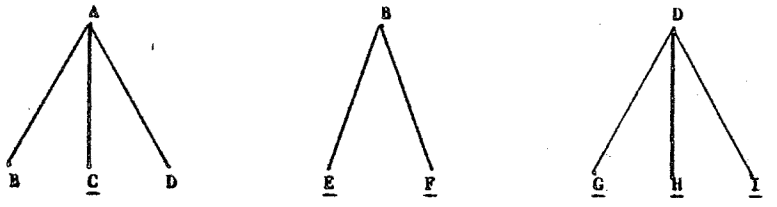
Ici, les commandes sont hiérarchisées, c'est-à-dire qu'elles peuvent être regroupées en un ensemle de menus (liste d'actions parmi lesquelles l'utilisateur peut choisir). A chaque menu correspond un environnement, environnement dans lequel les seules commandes activables par l'utilisateur sont celles du menu en question. Ainsi, l'utilisateur est très guidé par l'éditeur en ce qui concerne les actions qu'il peut effectuer.

Nous décrivons l'ensemble des menus par un arbre. Pour des raisons de lisibilité, cet arbre est divisé en plusieurs sous-arbres, chacun représentant un menu. Le nom du menu est donné par le nom attaché à la racine et les composants du menu sont tous les noms associés aux fils de la racine. Les fils soulignés sont des feuilles de l'arbre complet, c'est-à-dire qu'ils sont des actions terminales et qu'aucun sous-arbre ne leur est attaché.

Par exemple:



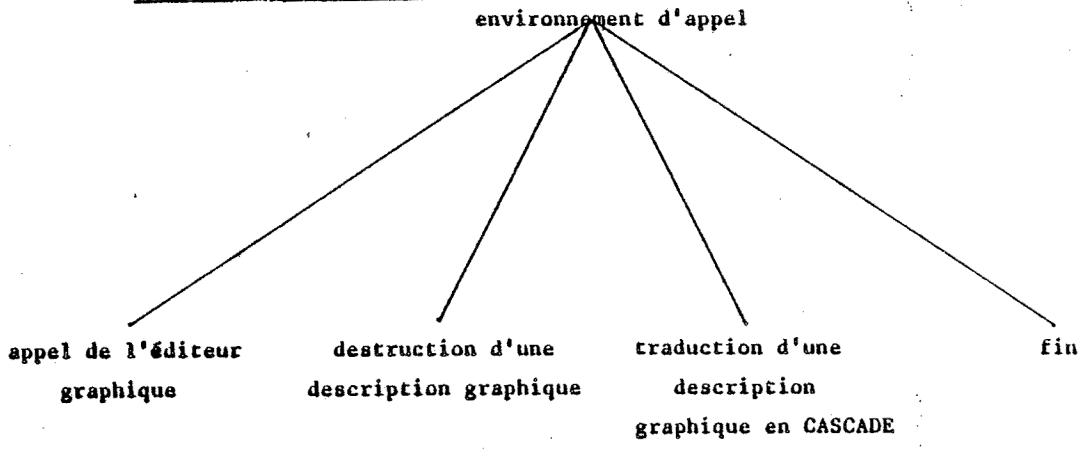
sera décrit par:



Nous voyons qu'il existe un sous-arbre attaché à B et à D qui sont donc des menus, alors que C, E, F, G, H, I sont des actions terminales.

Les menus de l'éditeur graphique de CASCADE:

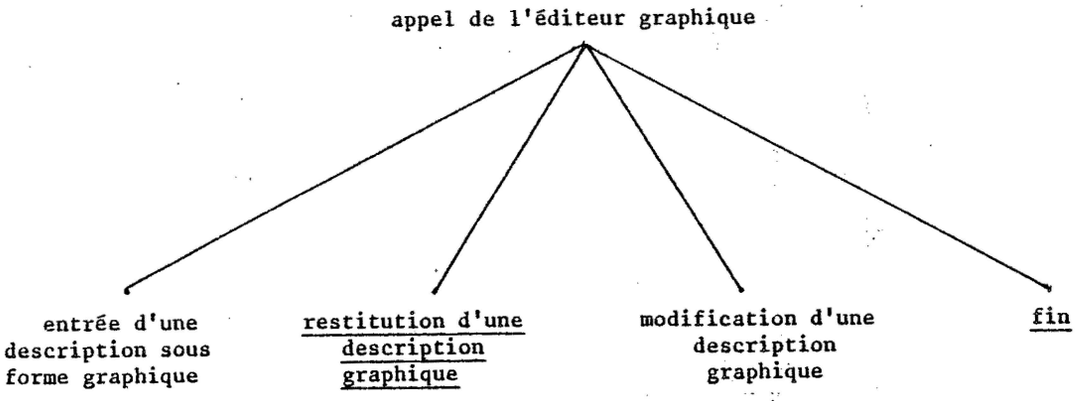
a) Environnement d'appel (niveau système)



Au niveau le plus élevé, l'utilisateur peut:

- soit décrire ou modifier un objet graphique et pour cela il devra entrer dans l'éditeur graphique;
- soit détruire une description graphique ce qui se gère au niveau système par la destruction de fichiers graphiques associés;
- soit traduire une description graphique en texte CASCADE ce qui se gère également au niveau système en générant un nouveau fichier contenant le texte source CASCADE de l'objet graphique choisi.

b) Appel de l'éditeur graphique



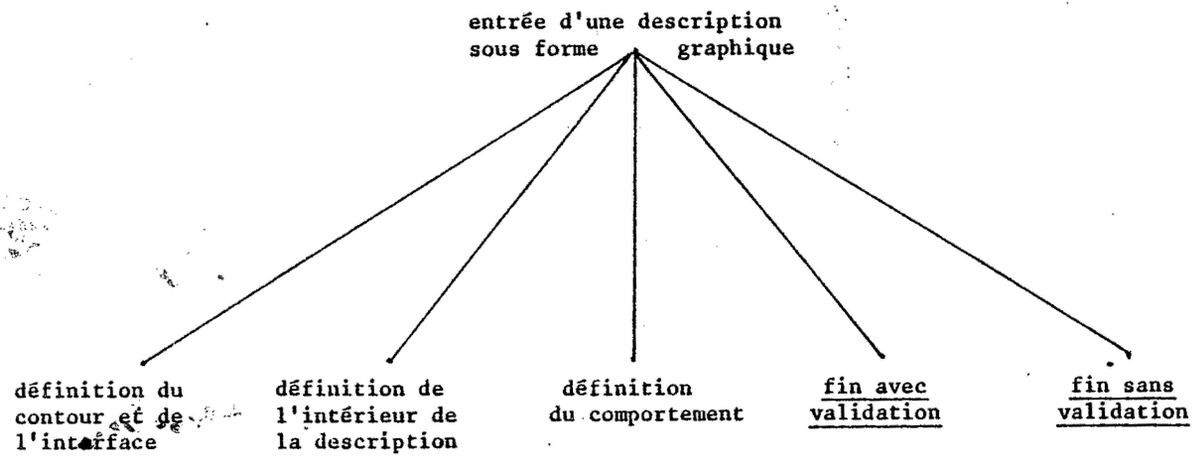
En entrant dans l'éditeur, l'utilisateur peut:

- soit entrer une description sous forme graphique, c'est-à-dire définir un nouvel objet graphique;
- soit restituer une description graphique déjà existante dans un fichier en vue de futures modifications;
- soit modifier une description graphique.

Il peut répéter ces trois commandes un certain nombre de fois. Lorsqu'il a fini, il sélectionne l'action "fin" qui a pour effet de le ramener au noeud père.

c) Entrée d'une description sous forme graphique

Ici, l'utilisateur est obligé de donner le nom de la description qu'il désire représenter sous forme graphique. Puis, le menu suivant lui est proposé:



Il peut donc:

- définir le contour et l'interface de sa description, ce qui lui permet d'associer une forme à son objet et de décrire les portes de communication de l'objet avec l'extérieur;

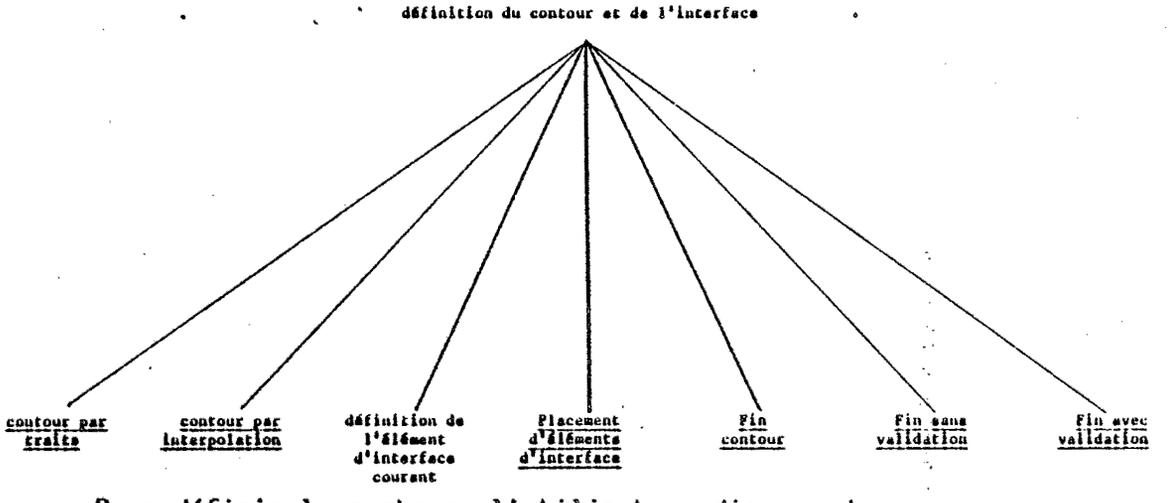
- définir l'intérieur de la description, c'est-à-dire placer les exemplaires de descriptions englobées et les connecter;

- définir le comportement de son objet (cette action ne sera jamais sélectionnée au niveau "portes logiques" par exemple);

- finir en validant ou non ce qu'il a défini, ce qui le ramène au niveau supérieur dans l'arbre.

d) Définition du contour et de l'interface

L'utilisateur est contraint de préciser ici le niveau de langage dans lequel il veut décrire son objet. Puis, il a le choix d'effectuer une des actions (ou plusieurs) associées aux commandes décrites dans le menu:



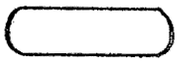
Pour définir le contour, l'utilisateur dispose de

- "contour par traits" qui permettra d'entrer un contour formé de segments de droite en donnant tous les points extrémités des segments;

- "contour par interpolation" qui permet d'entrer des parties de contour plutôt courbes;

- "fin contour" qui trace un segment entre le point où l'on se trouve et le premier point désigné.

L'utilisateur peut combiner les commandes "contour par traits" et "contour par interpolation" pour obtenir des contours mixtes:

 par exemple.

Cependant, il faut toujours que le point origine d'une nouvelle partie de contour soit le même que le point extrémité de la dernière partie de contour.

Notons que cela permet seulement de générer les contours que l'on peut tracer "sans lever le crayon de la feuille". Par exemple,

ne pourra pas représenter un objet, ce qui paraît logique, puisque structurellement ce schéma représente deux objets. Par contre,

peut parfaitement être le contour associé à un objet.

Pour définir l'interface de son objet, l'utilisateur dispose des deux commandes:

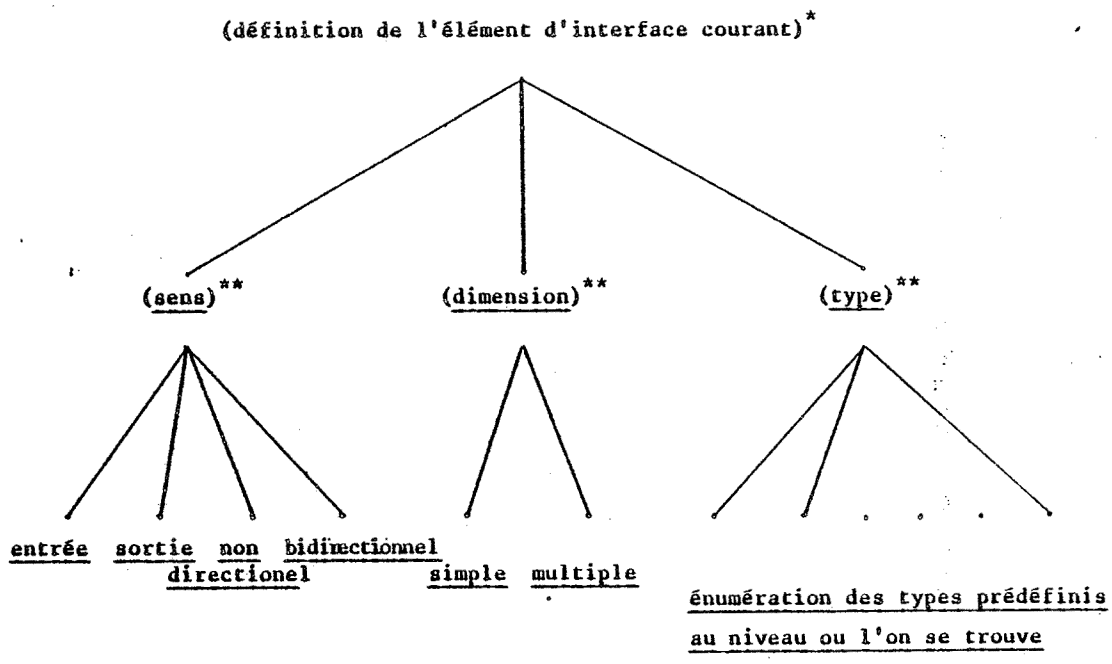
- "définition de l'élément d'interface courant" permet de définir le sens, la dimension (au sens nombre de sous-éléments) et le type de l'information attachés à l'élément d'interface;

- "placement d'élément d'interface" permet de placer sur le contour un élément d'interface ayant les caractéristiques de l'élément d'interface courant. Si on désire placer un élément d'interface dont l'une des caractéristiques au moins est différente de celles de l'élément d'interface courant, il faut d'abord redéfinir l'élément d'interface courant, puis placer un élément d'interface ayant les nouvelles caractéristiques en désignant son emplacement avec le réticule;

- enfin, "fin" avec ou sans validation permettent de remonter au niveau supérieur, une fois la définition du contour et de l'interface terminée.

e) Définition de l'élément d'interface courant

Pour définir l'élément d'interface courant, ce n'est pas un menu classique qui est présenté à l'utilisateur mais plutôt un ensemble de trois menus disjoints. Dans chacun de ces trois menus, il doit faire un choix et un seul, ce qui permettra de déterminer complètement l'élément d'interface courant.



( )* indique que l'on doit passer une fois et une seule par chacun des fils du noeud.

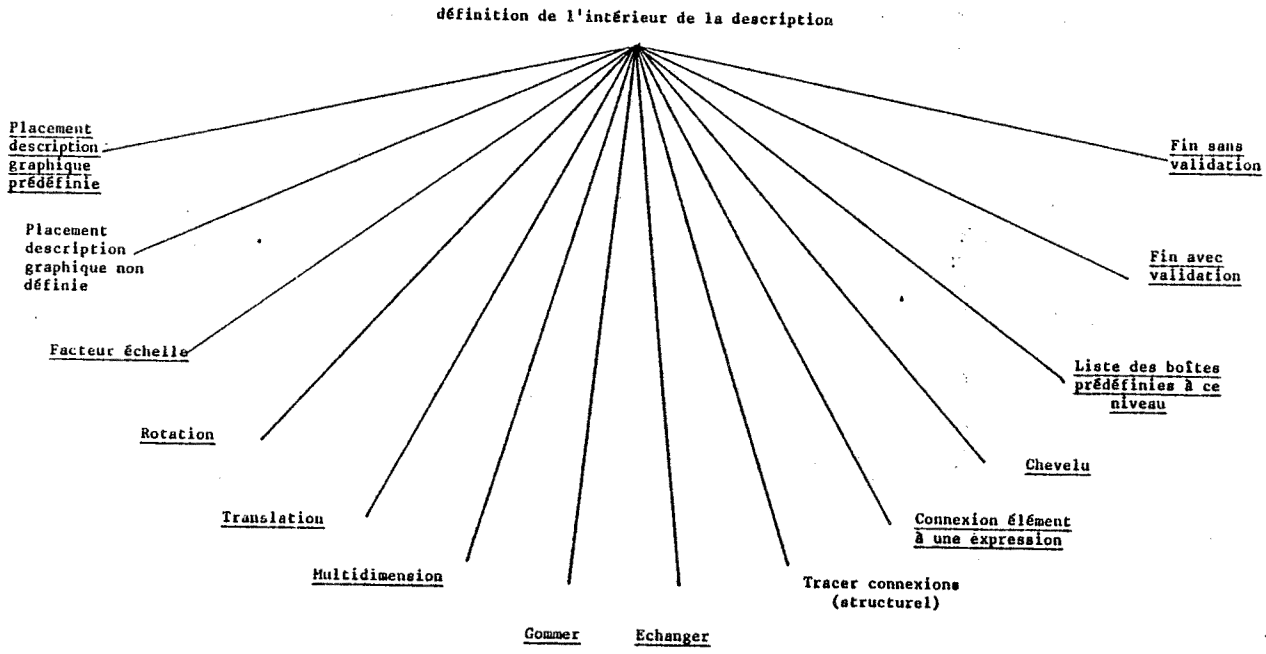
( )** indique que l'on doit choisir un fils et un seul et une seule fois. Si l'on sélectionne un menu ( )** une deuxième fois, il y a écrasement des informations précédemment entrées.

Dans le cas où la dimension est multiple, l'utilisateur donnera une liste de dimensions gauches et droites. Par exemple, [1:20] s'il désire un objet de vingt sous-éléments numérotés de 1 à 20.

Il précisera de plus si la représentation de l'objet est ou non compactée.

f) Définition de l'intérieur de la description

L'ensemble d'actions possibles est assez étendu dans cet environnement.



Construire l'intérieur d'une description consiste à placer des exemplaires de descriptions englobés à relier entre eux. Pour placer des exemplaires de descriptions externes, l'utilisateur dispose des deux comandes:

- "placement d'une description graphique prédéfinie" qui place un exemplaire de la description demandée en lui donnant un nom. Cette description a été définie auparavant par l'utilisateur ou fait partie des descriptions prédéfinies du niveau de langage auquel on se trouve;
- "placement d'une description graphique non définie" qui change l'environnement courant. L'utilisateur se trouve alors dans l'environnement "entrée d'une description sous forme graphique" et il définit ainsi la description inconnue graphiquement. Une fois cela achevé (fin de l'environnement "entrée d'une description sous forme graphique"), l'utilisateur se retrouve automatiquement dans l'ancien environnement. Il doit alors placer un exemplaire de la description qu'il vient de définir et lui donner un nom.

Si l'utilisateur se rend compte qu'il a mal placé ses boîtes, il dispose d'un ensemble de commandes qui lui permettent de corriger cela:

- "Facteur d'échelle" permet de modifier la taille d'une boîte, celle-ci étant arbitraire. La boîte devient alors plus importante (fe>1) ou plus petite (fe<1). De plus, l'utilisateur peut disposer de deux touches de fonction pour faire augmenter ou diminuer la taille de la boîte d'un facteur d'échelle prédéfini à chaque fois que l'on appuie sur l'une de ces touches.

- "Rotation" permet de modifier la position d'une boîte. Un exemplaire de description apparaît dans la position dans laquelle le modèle a été défini. Cela peut ne pas correspondre au désir de l'utilisateur. En montrant la boîte et en donnant un angle et un point de rotation, il verra le contour et les interfaces effectuer la rotation.

- "Translation" permet de déplacer des boîtes en donnant un vecteur de translation.

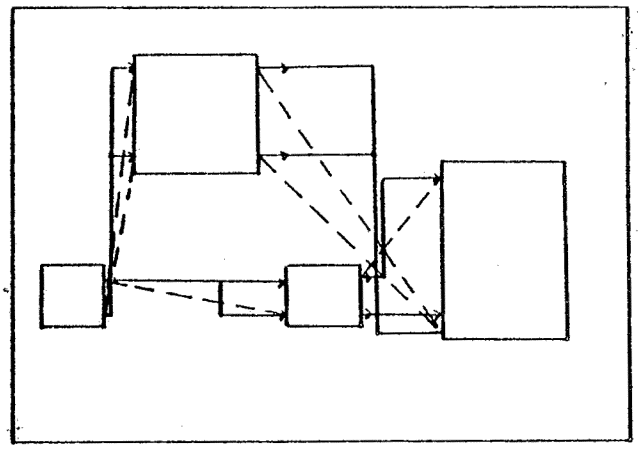
- "Gommer" permet d'effacer des boîtes placées par erreur.

- "Echanger" permet d'échanger la position de deux boîtes.

Pour voir si le placement de ses boîtes est judicieux, compte tenu du facteur de connectivité décrit en II.2.2, l'utilisateur dispose de la commande "chevelu" qui lui permet de doubler les connexions déjà tracées avec un trait reliant directement l'origine et l'extrémité de la connexion.

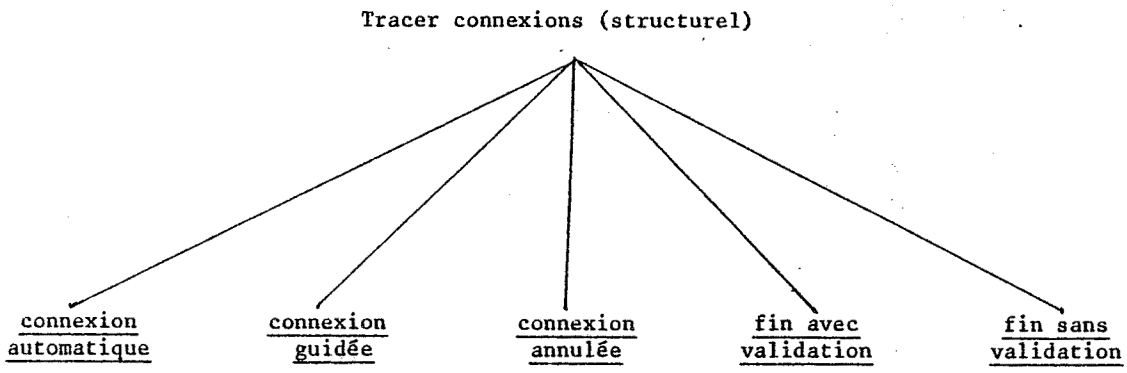
Exemple de l'utilisation de chevelu:

Soit le schéma:



En appelant la primitive "chevelu", les lignes en pointillé s'affichent et mettent en évidence un mauvais placement de la boîte du haut qui nécessiterait une translation.

g) Tracer connexions (structurel)

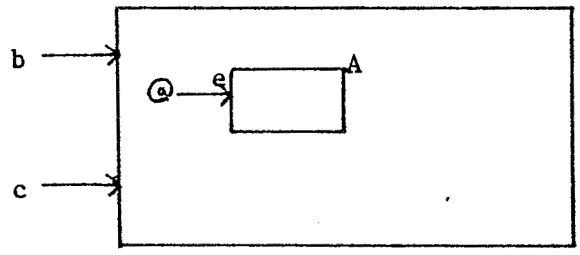


Pour connecter deux éléments d'interface entre eux, on fait appel à un nouveau menu car l'utilisateur peut soit tracer sa connexion en donnant point par point les points de rupture de sa connexion (connexion guidée), soit vouloir un tracé automatique entre ses deux points (connexion automatique). Si toutefois, la solution ne lui convient pas, il peut annuler la connexion et entrer sa propre solution. Il est également possible d'entrer le nom de variables locales (nommer la connexion). Le choix est laissé à l'utilisateur entre donner systématiquement un nom (éventuellement vide) à la connexion ou ne jamais en donner.

h) Connexion d'un élément à une expression

Le langage CASCADE permet dans certains cas d'associer à un élément d'interface une expression textuelle liée au comportement. Ceci est typiquement le cas où tout ne peut pas se représenter graphiquement. Il y aura alors un déroulement textuel pour que l'utilisateur puisse entrer l'expression. Un symbole particulier sera alors placé à l'extrémité de l'élément d'interface ainsi connecté.

Exemple:



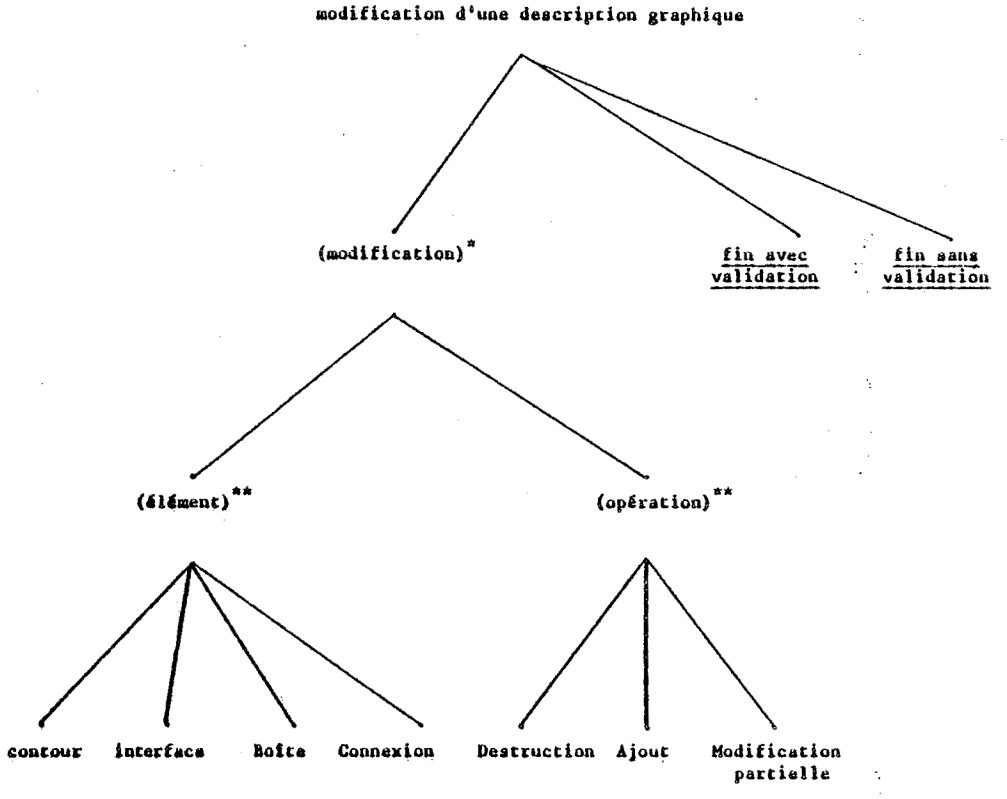
A.e est connecté à une expression entrée textuellement (par exemple si condition alors b + c, sinon b - c).

i) Définition du comportement

Suivant le niveau de langage auquel il se trouve, l'utilisateur entre le comportement de sa description graphiquement. Ceci fera l'objet du chapitre III.

j) Modification d'une description graphique

Cet environnement décrit toutes les modifications que l'on peut apporter à une description sous forme graphique. Les objets qui peuvent être modifiés sont le contour, les éléments d'interface, les boîtes englobées et les connexions. Les opérations que l'on peut effectuer sont destruction, ajout, modification partielle.



( )* et ( )** ont la même signification que dans l'environnement "définition de l'élément d'interface courant".

- "Destruction contour" détruit le contour et les interfaces de la description. L'environnement devient "définition du contour et de l'interface" pour remplacer le contour détruit.

- "Destruction interface" détruit tous les éléments d'interface désignés par l'utilisateur. Les connexions partant ou arrivant à un de ces éléments sont également détruits.

- "Destruction boîte" détruit la boîte et toutes les connexions à cette boîte.

- "Destruction connexion" détruit toutes les connexions montrées par l'utilisateur.

- "Ajout contour" n'a pas été mis en oeuvre.

- "Ajout élément d'interface" permet d'ajouter des éléments d'interface ayant les caractéristiques de l'élément d'interface courant.

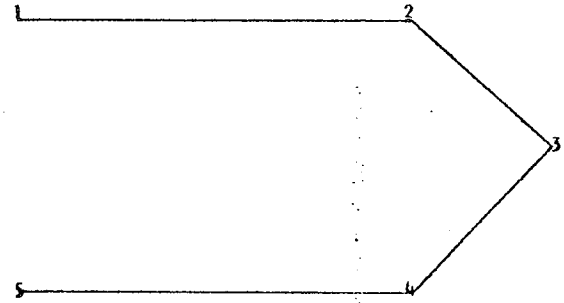
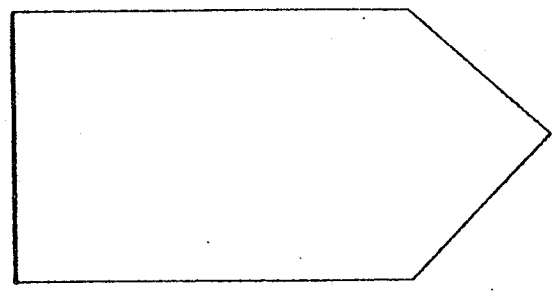
- "Ajout boîte" permet d'ajouter des exemplaires de boîtes prédéfinies ou non.

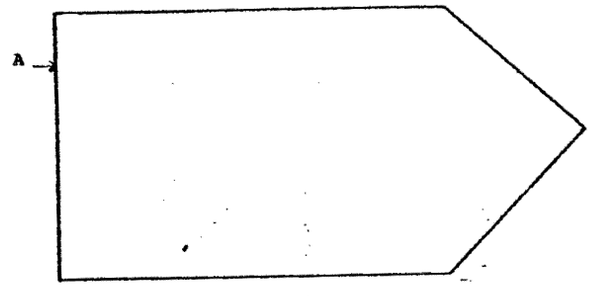
- "Ajout connexions" permet d'ajouter des connexions entre objets.
- "Modif contour" permet de changer le graphique de certaines parties du contour.
- "Modif interface" permet de changer une caractéristique de l'élément d'interface désigné.
- "Modif boîte" permet d'effectuer une translation, une rotation, d'appliquer un facteur d'échelle... sur une boîte englobée. Les connexions mises en jeu sont tracées par une ligne directe. L'utilisateur peut les retracer s'il le désire.
- "Modif connexion" permet de changer le tracé, la texture ou le compactage de la connexion.

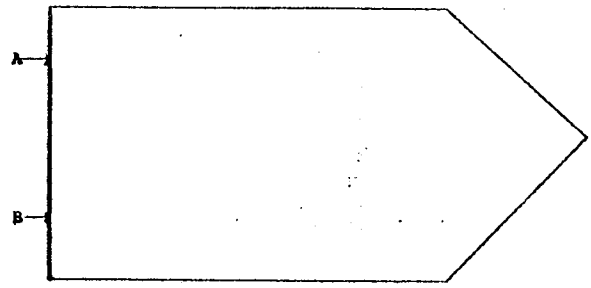
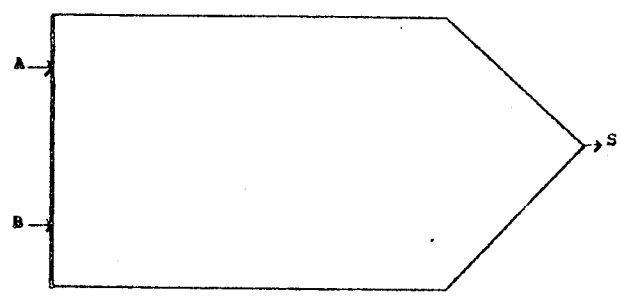
La description détaillée de toutes les commandes avec leur effet sur la structure de données graphique est donnée dans (Mar83b).

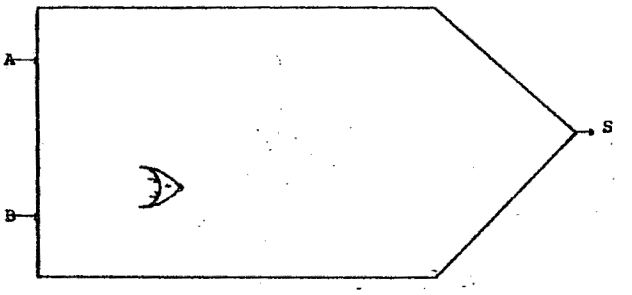
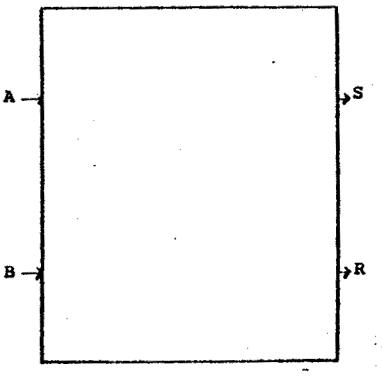
II.3.3 EXEMPLE D'UTILISATION D'EDICAS

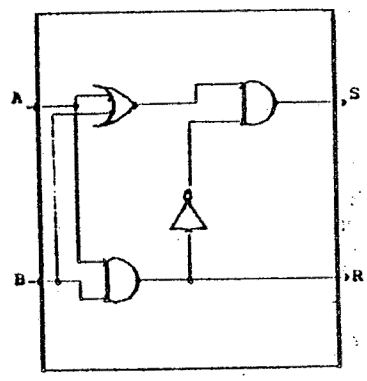
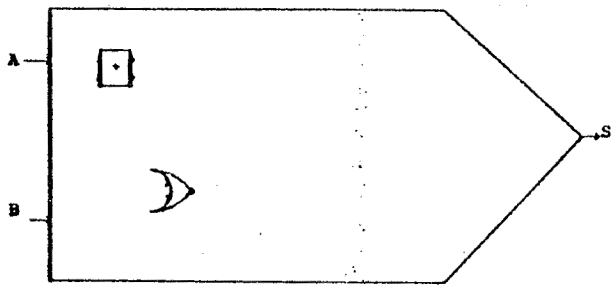
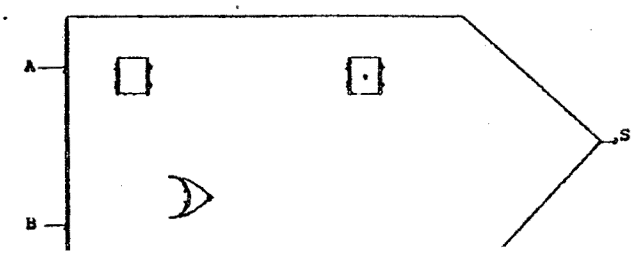
Pour construire l'objet graphique représentant l'additionneur série, nous allons passer par les étapes suivantes: initialement, nous nous trouvons dans l'environnement d'appel (environnement a).

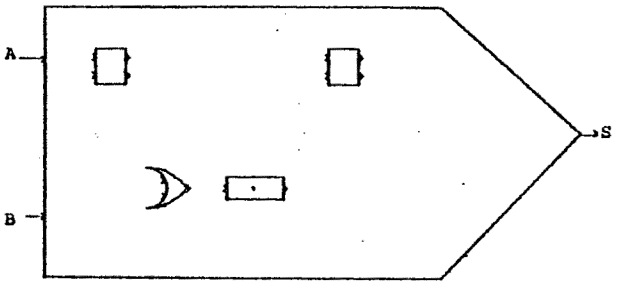
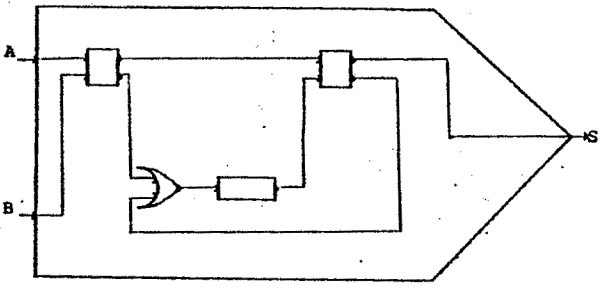
environnement	choix	écran
a)	Appel de l'éditeur graphique	
b)	Entrée d'une description sous forme graphique	
c)	Définition du contour et de l'interface	Message: Nom de la description ? ADDITIONNEUR
d)	Contour par traits	Message: Niveau de langage ? POLO Message: Entrer les points à relier 
	Fin contour	

environnement	choix	écran
d)	Définition de l'élément d'interface courant	
e)	Le choix pour les trois sous-menus donne sens = entrée dimensions = simple type = varlog	
d)	Placement d'un élément d'interface	Message: Nom de l'élément ? A Message: Entrer le point de placement  A est un élément d'entrée, de dimension simple et de type VARLOG (ce sont les caractéristiques de l'élément d'interface courant). (On se retrouve dans la même situation que précédemment et on désire à nouveau placer son élément)

environnement	choix	écran
d)	Placement d'un élément d'interface	Message: Nom de l'élément ? B Message: Entrer le point de placement  B a les mêmes caractéristiques que A
	Définition de l'élément d'interface courant	
e)	Le choix des trois sous-menus donne sens = sortie dimension = simple type = varlog	
d)	Placement d'un élément d'interface	Message: Nom de l'élément ? S Message: Entrer le point de placement 
	Fin avec validation	

environnement	choix	écran
c)	Définition de l'intérieur de la description	
f)	Placement d'une description graphique prédéfinie	Nom de la description ? OR Nom de l'exemplaire ? OR1 Donner le point de placement 
	Placement d'une description graphique non définie	
c)	Définition du contour et de l'interface . . . On obtient	Nom de la description ? ADD . . . 

environnement	choix	écran
. . .	Définition de l'intérieur . . . On obtient On valide et on se retrouve dans l'environnement f)	. . . 
f)	Placement d'une description graphique prédéfinie	Message: Nom de l'exemplaire ? ADD1 Message: Entrer le point de placement  Nom de la description ? ADD (il est prédéfini à présent) Nom de l'exemplaire ? ADD2 Entrer le point de placement 

environnement	choix	écran
	Placement d'une description graphique prédéfinie (on suppose que TETA est déjà définie)	Nom de la description ? TETA Valeur de délai ? 15 (on fixe le nombre d'intervalles de temps entre deux arrivées) (délai est un attribut de la boîte TETA) Nom de l'exemplaire ? TETA1 Donner le point de placement 
. . .	Tracer connexions . . .	. . .
	On obtient 	
	On valide successivement et on sort de l'éditeur avec un nouvel objet	

II.3.4 CLASSIFICATION DES COMMANDES D'EDICAS

En suivant les critères adoptés en II.3.1, les actions terminales d'EDICAS peuvent être ainsi classifiées.

Groupe G1: Affichage seulement

- "Chevelu" qui n'ajoute que des traits destinés à aider momentanément l'utilisateur.
- "Liste des boîtes prédéfinies".
- Des fonctions non mentionnées dans les menus qui permettent à tout moment de faire sortir une copie claire de l'écran en ajoutant le nom de l'utilisateur, la version du dessin...

Groupe G2a: Influence sur l'objet dessiné mais pas de sémantique associée à la Transformation

- "Restitution description graphique".
- "Rotation, translation, facteur d'échelle".
- "Echanger deux boîtes".

Groupe G2b: Influence de la tranformation sur la sémantique

- "contour par traits, par interpolation".
- "Placement d'éléments d'interface".
- "Fin contour".
- "Multidimension".
- "Connexions automatiques, guidées, annulées".

Groupe G3: Influence sur les segments contenant les structures graphiques des descriptions

- "Destruction description graphique".
- "Traduction graphique ---> CASCADE".
- "Fin avec validation" de la définition d'un élément graphique.

II.4 TRADUCTION EN LANGAGE CASCADE

La traduction d'une description graphique en langage source CASCADE s'appuie en grande partie sur la structure de données graphique. Cette structure graphique est construite ou modifiée dans l'éditeur EDICAS lorsque l'on crée ou modifie une description graphique. Avant l'étude de la traduction en CASCADE, exposons la structure graphique attachée à une description.

II.4.1 STRUCTURE GRAPHIQUE ATTACHEE A UNE DESCRIPTION

Une description est représentée par un ensemble de champs:

NOM	NL	CONT	INT	BTE	CONN	ATT	COMP

Une description graphique est définie par:

- Son nom (chaîne de caractères) (NOM).
- Son niveau de langage (chaîne de caractères) (NL).
- Le contour graphique qui lui est associé (CONT). La représentation de ce contour est donnée par CONT qui est un pointeur sur un ensemble de parties de contour.
- Son interface (INT). La représentation en est donnée par INT qui est un pointeur sur un ensemble d'éléments d'interface.
- Les boîtes imbriquées qu'elle contient. La représentation de ces boîtes est donnée par BTE qui pointe sur un ensemble de boîtes imbriquées.
- Les connexions qui existent entre les différentes boîtes imbriquées. C'est CONN qui les représente en pointant sur un ensemble de connexions.
- Les attributs qui lui sont attachés si cette description décrit une famille de modèles. ATT pointe sur un ensemble d'attributs.
- Le comportement qu'elle vérifie (COMP). Cela ne sera pas considéré dans cette partie.

a) Représentation interne du contour

La représentation interne du contour ne nous intéresse pas ici puisque le contour n'aura aucune influence sur la traduction en CASCADE. Cette représentation peut être trouvée dans (Mar83b).

b) Représentation interne des éléments d'interface

Un élément d'interface sera défini par:

NOM	SENS	DT	LD	C	X1	Y1	XN	YN	NATCON	LPC	EXP	SUIV

En effet, un élément d'interface est entièrement déterminé si on connaît:

- Son nom. (NOM)
- Son sens (entrée, sortie, bidirectionnel, non directionnel). (SENS)
- Son descripteur de type. (DT)
- Sa liste de dimensions représentée par LD qui pointe sur une liste de dimensions gauches et droites. (Représentation d'une telle liste: DG DD SUIV)
- Si la représentation est compactée ou non. (C)
- Les coordonnées de l'élément d'interface. (X1, Y1)
- Les coordonnées de "la fin" de l'élément d'interface si l'élément n'est pas compacté. (XN, YN)
- La nature de la connexion. (NATCON)
  - connexion à un autre élément d'interface
  - connexion à une expression
  - non connecté
- La liste de toutes les connexions dans lesquelles cet élément d'interface est impliqué. (LPC)
- L'expression textuelle à laquelle il est connecté s'il s'agit d'une connexion à une expression. (EXP)

c) Représentation interne des boîtes imbriquées

Une boîte imbriquée sera représentée par la liste de champs:

NEX	NMO	X1	Y1	XN	YN	FEC	ARO	LD	C	SUIV

car une boîte imbriquée est totalement définie par la donnée de

- Son nom d'exemplaire. (NEX)
- Le nom de son modèle (NMO). Ici, la représentation est un pointeur sur une copie du modèle dont les attributs ont été fixés.

- Les coordonnées de son barycentre. (X1, Y1)

- Les coordonnées de la nième boîte dans le cas de non compactage. (XN, YN)

- Son facteur d'échelle. (FEC)

- Son angle de rotation (ARO) par rapport à sa position d'origine.

- Sa liste de dimensions représentée par LD qui pointe une liste de dimensions gauches et droites.

- Si la représentation est compactée ou non. (C)

d) Représentation interne des connexions

Une connexion est décrite par:

LPR	SOL	SUIV

- La liste des points à relier. (LPR) (Un point à relier est défini par:

ND	NEL	SUIV

  - Le nom de l'exemplaire de description contenant l'élément d'interface à relier. (ND)
  - Le nom de l'élément d'interface à relier. (NEL))

- La solution (SOL) qui est constituée des points reliant les éléments etre eux (elle est entrée soit par l'utilisateur dans le cas d'une connexion guidée, soit automatiquement par l'algorithme de routage).

e) Représentation interne des attributs

Un attribut est décrit par:

NOM	TYPE	VE	SUIV

- Son nom. (NOM)

- Son type. (TYPE)

- Sa valeur effective (VE) si elle a été fixée (boîte englobée).

Une description plus détaillée de la structure et l'influence de chaque commande d'EDICAS sur cette structure est décrite dans (Mar83b). La description de la structure que nous avons donnée suffit cependant à générer le texte CASCADE correspondant à une description.

II.4.2 TRADUCTION A PARTIR DE LA STRUCTURE INTERNE D'UNE DESCRIPTION

A partir de la représentation interne d'une description graphique, nous cherchons à avoir un texte source correspondant. Une certaine fonction nous fait donc passer d'une représentation interne sous forme de listes à un texte formé d'une concaténation de chaines de caractères. Les chaînes de caractères constantes seront mentionnées entre " et ". La chaîne vide est notée "Ɛ". L'opération de concaténation est notée "//". La notation pointée "." permet de sélectionner un des champs de l'une des structures.

Description pointe sur la structure de la description à traduire.

Soit une fonction de transformation Ø vérifiant :

```
Ø (description) = "lanref" // description.nl //
                  "description" // description.nom //
                  Ø1(description.att) //
                  Ø2(description.int) //
                  "corps" // Ø3(description.bte)
                          // Ø4(description.bte) // "fin"
```

où Ø1 génère les attributs à partir de la liste des attributs de la structure, Ø2 génère l'interface à partir de la liste des éléments d'interface, Ø3 génère la partie "externe"..., et Ø4 génère la liste des unités.

Cette traduction n'est valable qu'au niveau "POLO" puisqu'il n'y a pas de partie comportement à ce niveau.

Etudions chacune des fonctions Øi, i = 1 à 4.

liste_att pointe une liste d'attributs (structure définie en II.4.1.e).

```
Ø1(liste_att) = si liste_att vide alors Ɛ
                sinon "(" // Ø11(liste_att) // ")"



avec Ø11(liste_att) = liste_att.type // liste_att.nom
                      // (si liste_att.suiv vide alors Ɛ
                      sinon ";" // Ø11(liste_att.suiv))
```

Nous ne générons des parenthèses que si la liste n'est pas vide (c'est ce que fait Ø1) et nous mettons entre les parenthèses la liste des attributs avec leurs types, séparés par des ; (Ø11 qui s'appelle récursivement)

liste_int pointe une liste d'éléments d'interface (structure définie en II.4.1.b).

```
Ø2(liste_int) = si liste_int vide alors ε
                sinon "(" // Ø21(liste_int) // ")"


avec Ø21(liste_int) = liste_int.sens // liste_int.dt //
                liste_int.nom // Ø22(liste_int.ld) //
                                (si liste_int.suiv vide alors ε
                                sinon ";" // Ø21(liste_int.suiv)
```

liste_dim pointe une liste de dimensions (structure définie en II.4.1.b).

```
et Ø22(liste_dim) = si liste_dim vide alors ε
                    sinon `[' // Ø221(liste_dim) // `]'


  Ø221(liste_dim) = liste_dim.dg // ":" // liste_dim.dd
                    // (si liste_dim.suiv vide alors ε
                    sinon ";" // Ø221(liste_dim.suiv))
```

Nous ne générons des parenthèses que si la liste des éléments d'interface n'est pas vide (c'est ce que fait Ø2). Si c'est le cas, nous énumérons chaque élément d'interface précédé par son sens et son type et postfixé s'il y a lieu par ses dimensions. Ø21 assure l'énumération de tous les éléments en s'appelant récursivement et Ø22 génère des crochets s'il existe des dimensions. Si c'est le cas, Ø221 énumère ces dimensions.

liste_bte pointe sur une liste de boîtes imbriquées (structure définie en II.4.1.c).

```
Ø3(liste_bte) = si liste_bte vide alors ε
                sinon "externe" // Ø31(liste_bte) // ";"


avec Ø31(liste_bte) = (si déjà_cité(liste_bte.nmo.nom)
                      alors ε sinon (liste_bte.nmo.nom))
                      // (si liste_bte.suiv vide alors ε
                          sinon Ø31(liste_bte.suiv))
```

Ø3 génère à partir de la liste des boîtes englobées la partie déclaration d'externe. S'il existe des boîtes englobées, le mot clé "externe" est généré et Ø31 assure l'énumération de tous les noms de modèles utilisés en s'assurant de ne pas les répéter dans le cas où plusieurs boîtes feraient référence au même modèle.

```
Ø4(liste_bte) = si liste_bte vide alors ε
                sinon "unites" // Ø41(liste_bte)
```

```
avec Ø41(liste_bte) = liste_bte.nmo.nom // Ø42(liste_bte.nmo/att)
                      // liste_bte.nex // Ø43(liste_bte.nmo.int)
                      // (si liste_bte.suiv vide alors ε
                          sinon ";" // Ø41(liste_bte.suiv))

     Ø42(liste_att) = si liste_att vide alors ε
                      sinon "(" // Ø421(liste_att) // ")"

     Ø421(liste_att) = liste_att.ve // (si liste_att.suiv vide alors ε
                                sinon "," // Ø421(liste_att.suiv))

     Ø43(liste_int) = si liste_int vide alors ε
                      sinon "(" // Ø431(liste_int) // ")"

     Ø431(liste_int) = (si liste_int.pco vide alors ε
                       sinon (si reference.avant(liste_int.pco) alors ε
     sinon (si liste_int.pco.lpr.autre element.nd
                              = description englobante
            alors liste_int.pco.lpr.autre element.nel.nom
            sinon liste_int.pco.lpr.autre element.nd.nom
               // "." // liste_int.pco.lpr.autre element.nel.nom))
               // (si liste_int.suiv vide alors ε
                   sinon "," // Ø431(liste_int.suiv))
```

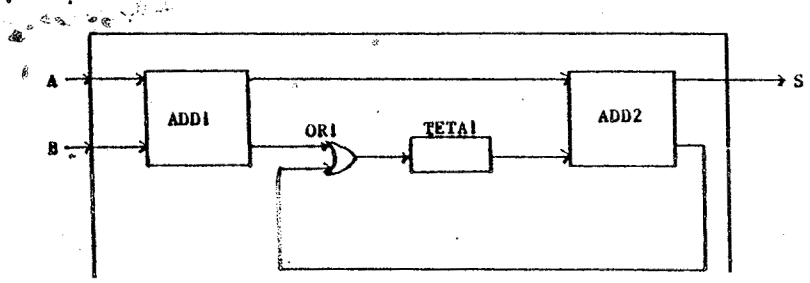
Remarque: autre élément est une facilité d'écriture pour indiquer l'autre élément d'interface mis en jeu dans la connexion.

Ø4 génère le mot clé "unités" dans le cas où il y a des exemplaires de descriptions. Ø41 énumère ces exemplaires en donnant le nom du modèle, la liste des valeurs des attributs si le modèle décrit une famille de descriptions (Ø42), le nom de l'exemplaire et les connexions avec notation pointée (Ø43). Ø431 gère l'énumération de toutes les connexions en ne générant rien si l'élément d'interface n'est pas connecté ou s'il y a une référence avant.

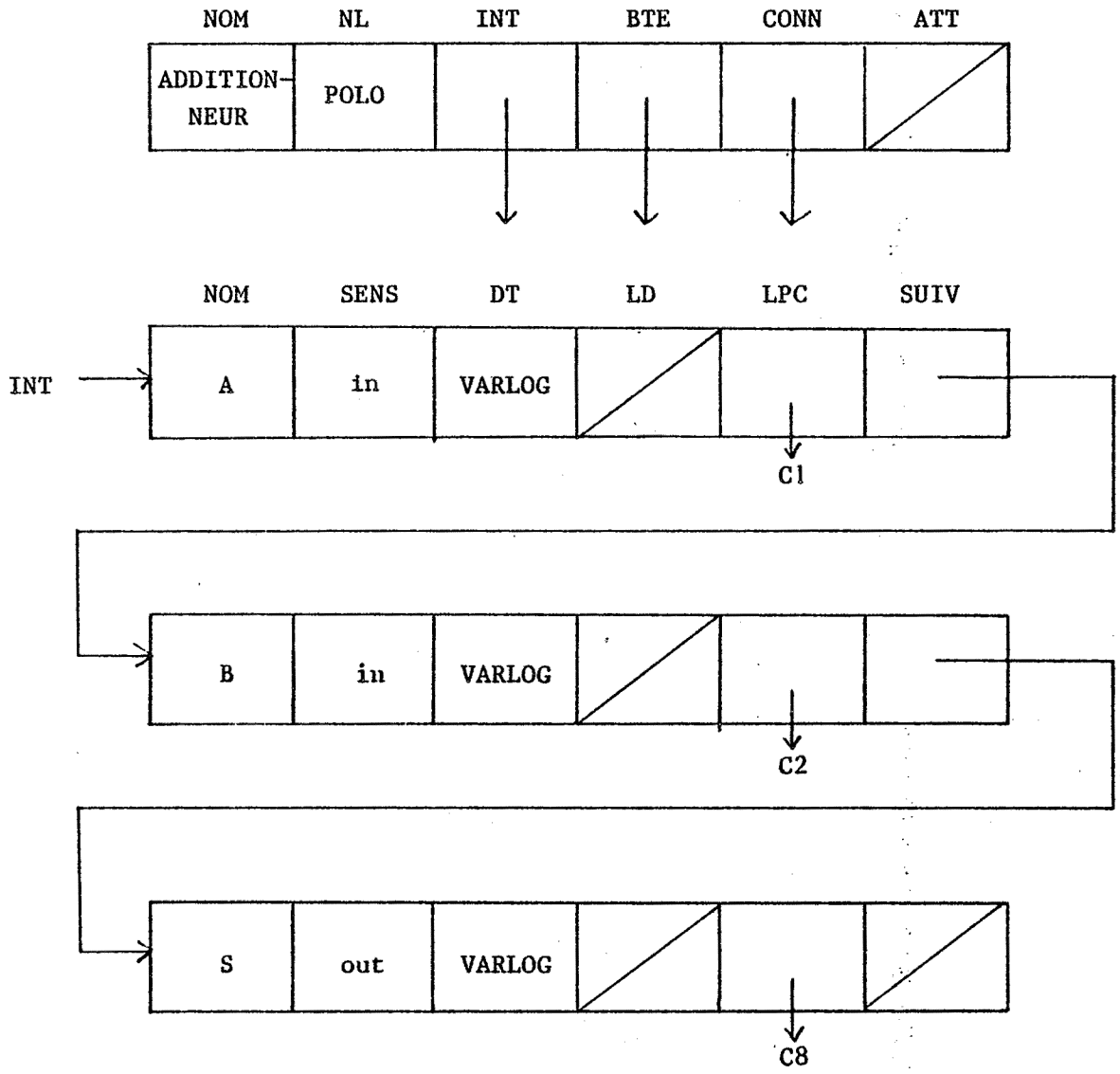
Nous allons donner un exemple de traduction, ce qui aidera la compréhension des formules données ci-dessus.

II.4.3 EXEMPLE DE TRADUCTION

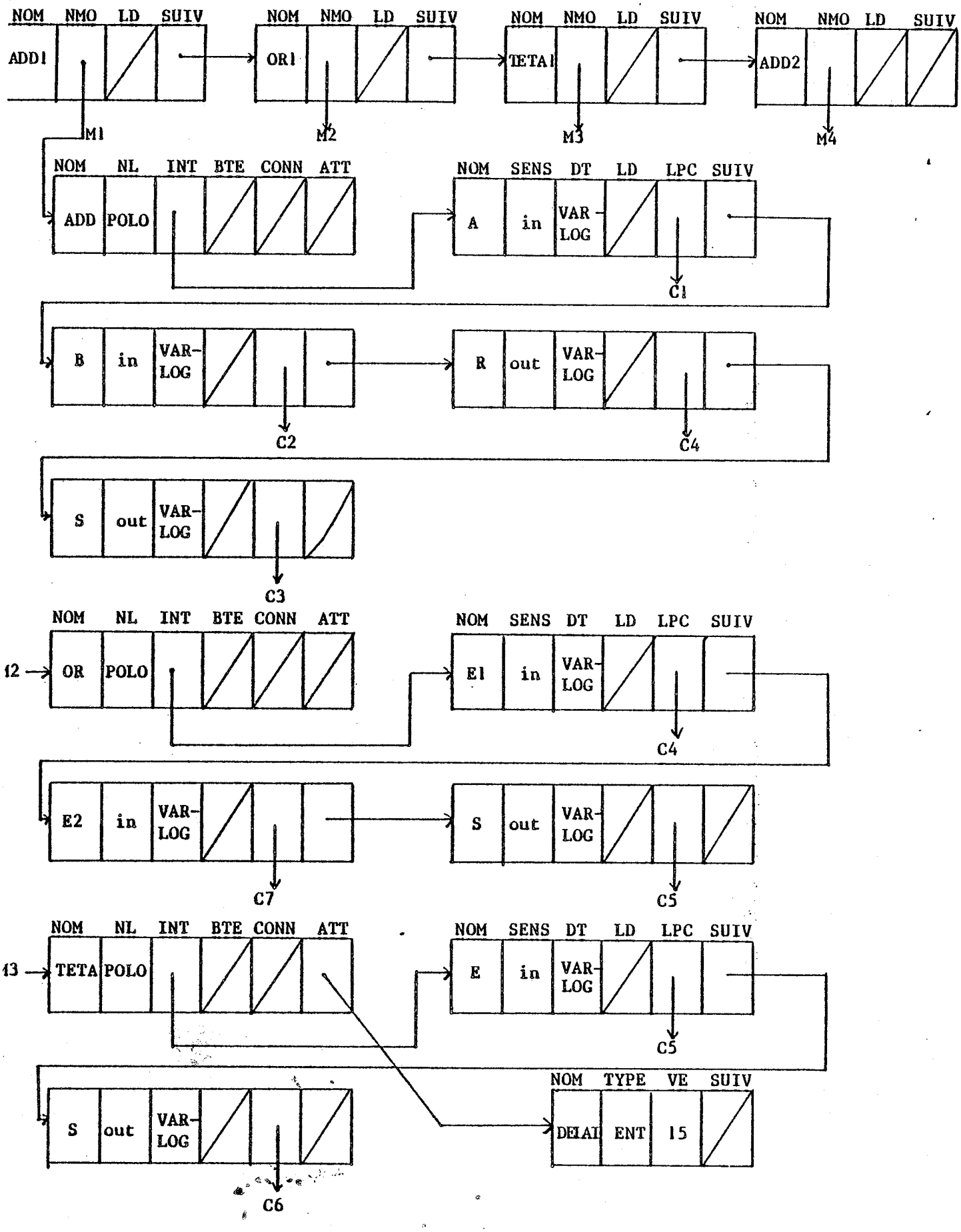
Reprenons l'exemple de l'additionneur série. Nous nous proposons, à partir du schéma et de la représentation graphique associée de l'additionneur ADDITIONNEUR, de trouver la description CASCADE équivalente.



La représentation interne de cette description est la suivante: (on ne représente que les champs ayant un intérêt pour la traduction: tous les champs purement graphiques ont été supprimés pour l'exposé de cet exemple).

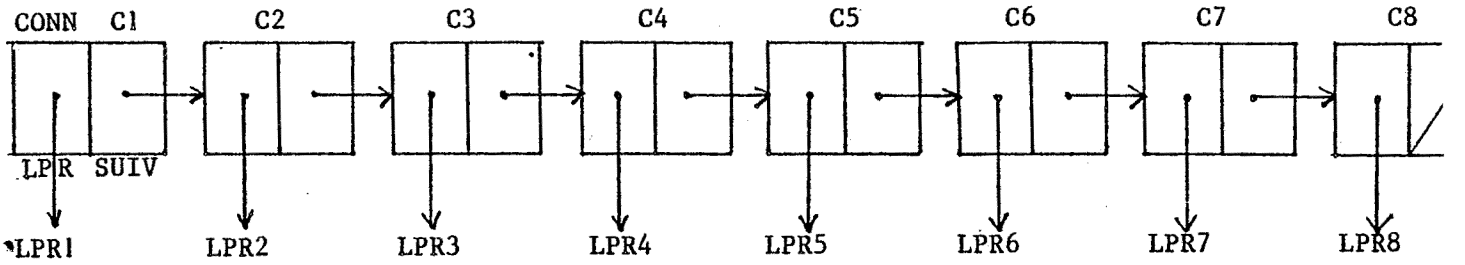


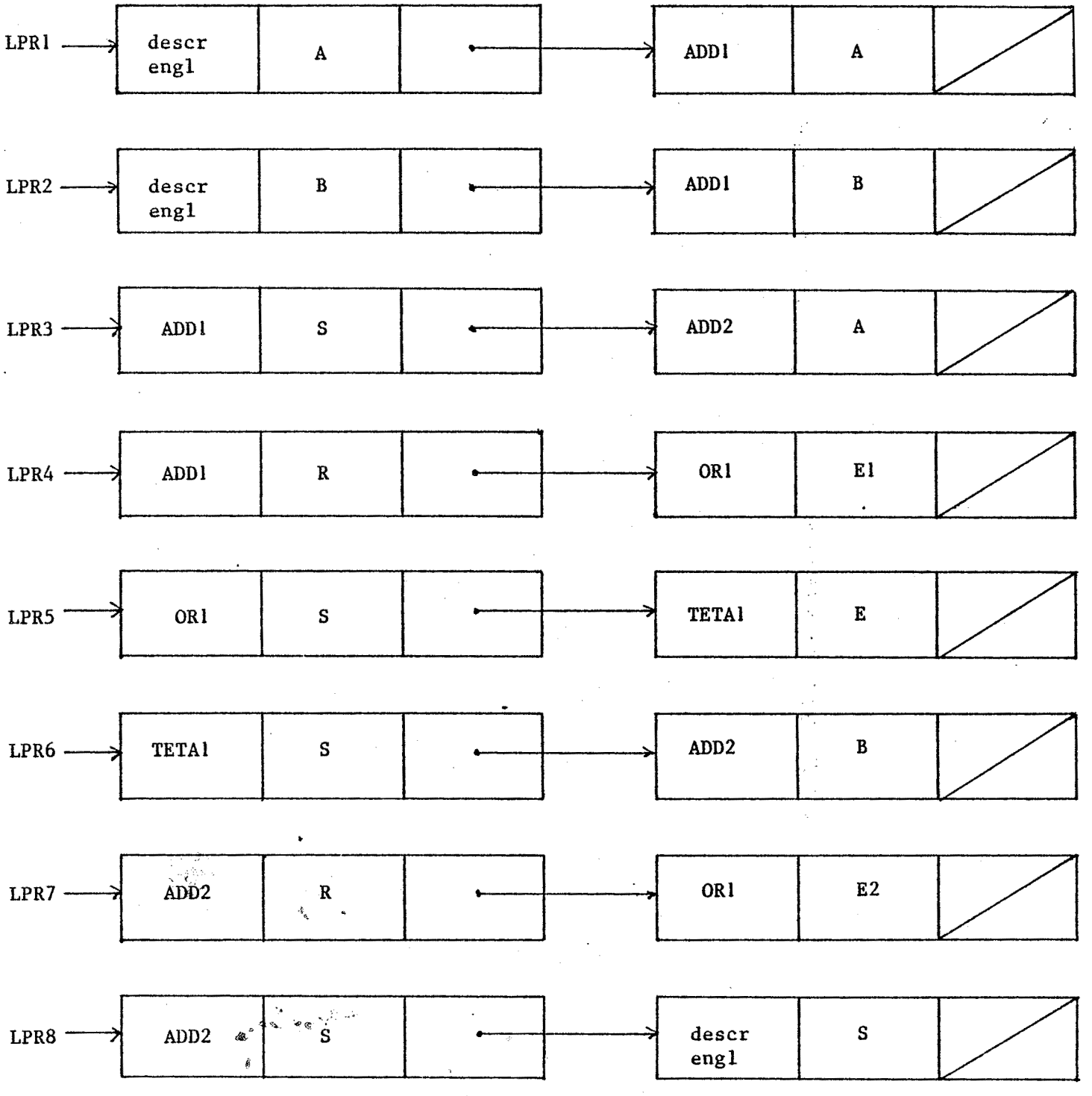
(description de l'interface)



(description btes imbriquées)

M4 n'est pas décrit mais est similaire à M1.


CONN
C1
C2
C3
C4
C5
C6
C7
C8
LPR
SUIV
LPR1
LPR2
LPR3
LPR4
LPR5
LPR6
LPR7
LPR8


LPR1
descr engl
A
ADD1
A
LPR2
descr engl
B
ADD1
B
LPR3
ADD1
S
ADD2
A
LPR4
ADD1
R
OR1
E1
LPR5
OR1
S
TETA1
E
LPR6
TETA1
S
ADD2
B
LPR7
ADD2
R
OR1
E2
LPR8
ADD2
S
descr engl
S

Appliquons la fonction Ø à la structure ADDITIONNEUR

```
Ø(ADDITIONNEUR) = lanref POLO description ADDITIONNEUR //
                    Ø1(ADDITIONNEUR.ATT) //
                    Ø2(ADDITIONNEUR.INT) //
                    corps //
                        Ø3(ADDITIONNEUR.BIE) //
                        Ø4(ADDITIONNEUR.BIE) //
                    fin

Ø1(ADDITIONNEUR.ATT) = Ɛ

Ø2(ADDITIONNEUR.INT) = ( // Ø21(ADDITIONNEUR.INT) // )

Ø21(ADDITIONNEUR.INT) = in VARLOG A // Ø22(ADDITIONNEUR.INT.LD)
                                // ;  Ø21(ADDITIONNEUR.INT)
```

en remplaçant, on obtient

```
Ø21(ADDITIONNEUR.INT) = in VARLOG A;  in VARLOG B;
                            out VARLOG S
```

d'où

```
Ø2(ADDITIONNEUR.INT) = (in VARLOG A;  in VARLOG B;
                                out VARLOG S)
```

En appliquant les fonctions associées à Ø3, on obtient

```
Ø3(ADDITIONNEUR.BIE) = externe ADD,OR,IETA;
```

Enfin Ø4(ADDITIONNEUR.BIE) = unités // Ø41(ADDITIONNEUR.BIE)

```
Ø41(ADDITIONNEUR.BIE) = ADD //Ɛ// ADD1
                            // Ø43(ADDITIONNEUR.BIE.NMO.INT)
                            // ;  // Ø41(ADDITIONNEUR.BIE.SUIV)

Ø43(ADDITIONNEUR.BIE.NMO.INT) = (// Ø431(ADDITIONNEUR.BIE.NMO.INT) //)

Ø431(ADDITIONNEUR.BIE.NMO.INT) = A, B, //Ɛ//, //Ɛ
```

Ainsi, de proche en proche, on obtient

```
Ø4(ADDITIONNEUR.BIE) =

            unites ADD ADD1(A, B,,);
                   OR OR1 (ADD1.R,,);
                   IETA(15) IETA1 (OR1.S,);
                   ADD ADD2(ADD1.S, IETA1.S, OR1.E2,S)
```

D'où

```
Ø(ADDITIONNEUR) = lanref POLO description ADDITIONNEUR
                    (in VARLOG A;  in VARLOG B;  out VARLOG S)
                  corps
                      externe ADD, OR, TETA;
                      unites ADD ADD1(A, B,,);
                             OR OR1(ADD1.R,,);
                             TETA(15) TETA1(OR1.S,);
                             ADD ADD2(ADD1.S, TETA1.S, OR1.E2, S)
                  fin
```

Pour des raisons de clarté de l'exposé, nous avons omis tous les caractères blancs et les retours chariot générés pour la mise en page de la description textuelle. Cette mise en page est cependant effectuée par l'outil.

II.5 PASSAGE DU TEXTE CASCADE A UNE REPRESENTATION GRAPHIQUE CORRESPONDANTE

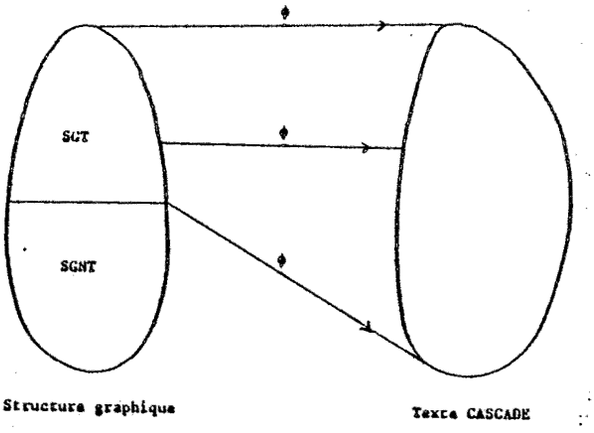
Nous avons deux représentations possibles d'une description: une graphique, l'autre textuelle. Ces deux descriptions ne sont pas totalement équivalentes. Nous avons étudié sans difficulté le passage d'une description graphique à une description textuelle. Pour faire cela, nous n'utilisons qu'une partie de la structure graphique. Toutes les informations nécessaires à cette traduction font partie de la structure graphique. Il existe cependant d'autres parties de la structure graphique qui ne sont pas utilisées pour cela (la description du contour par exemple).

La structure graphique est donc constituée de deux parties:

- celle qui ne sert pas pour la traduction (SGNT);
- celle qui est utilisée pour générer du texte CASCADE (SGT).

On peut déduire qu'à partir d'un texte CASCADE, il est possible de retrouver SGT de la représentation interne graphique alors que l'on ne peut pas retrouver SGNT.

Cela s'illustre par



On voit que par une réciproque $\emptyset^{-1}$, on arrive à retrouver SGT. Le contour de la description graphique est, par exemple, un élément qui appartient à SGNT. Il est évident qu'à partir de la description textuelle, on ne pourra pas associer un contour automatiquement (ou alors de manière arbitraire).

La conclusion de cela est que pour la définition graphique d'une description, il faut, en plus du texte CASCADE, établir un dialogue avec l'utilisateur qui lui permette de préciser la partie SGNT.

Scénario de définition d'un objet graphique à partir du texte CASCADE

Nous présentons ici un dialogue possible avec l'utilisateur. Lors de la transformation d'un texte CASCADE d'une description D en une description graphique, la première chose à préciser par l'utilisateur est le contour associé à D. Cela peut être fait de la même manière que lorsque l'utilisateur entre sa description graphique directement (voir menu correspondant). Une fois le contour décrit, la liste d'éléments d'interface s'affiche dans une fenêtre prévue à cet effet. L'utilisateur désigne alors les emplacements sur le contour où il désire que ces éléments soient placés. Toutes les autres informations concernant les éléments d'interface (sens, type...) sont déjà connues.

La liste des exemplaires de description s'affiche alors. De même, l'utilisateur doit préciser l'emplacement de ces boîtes. Il peut de plus utiliser les fonctions de transformation (facteur d'échelle, rotation...) pour obtenir exactement le dessin désiré. Ceci laisse supposer que les descriptions dont on a pris des exemplaires sont déjà définies graphiquement. Si ce n'est pas le cas, on applique le même processus décrit ici pour ces boîtes.

Enfin, les connexions devront être tracées. Comme les points à relier sont connus, nous proposons de les mettre en évidence (surbrillance) et l'utilisateur les reliera soit par l'algorithme de routage, soit par connexion guidée.

Nous n'avons ainsi donné que des informations graphiques et nous remarquons que tous les champs de la structure que nous avons remplis sont exactement ceux qui ne servent pas à la traduction par Ø (les éléments de SGNT).

Nous obtenons ainsi toute la structure graphique correspondant à la description D.

II.6 REALISATION. RESULTATS ET EVALUATIONS

Pour valider nos idées, surtout en ce qui concerne la traduction d'une description graphique en texte CASCADE, nous avons réalisé un petit prototype ne fonctionnant qu'au niveau structurel et qui réalise à peu près les fonctions décrites dans tout ce chapitre. Un manuel d'utilisation est disponible (Mar84b).

La réalisation de ce prototype s'est faite sur matériel VAX780, sous système VMS. Le programme a été écrit en langage PASCAL. Le matériel graphique dont nous avons disposé pour la mise au point a été un TEKTRONIX 4114 et le logiciel graphique de base utilisé a été CLOVIS.

Le programme a été réalisé de manière modulaire car bien que la norme du langage PASCAL ne permette pas la compilation séparée, la majorité des compilateurs PASCAL offrent la possibilité aux programmeurs d'utiliser cette notion. C'est ce que nous avons fait.

Nous avons déjà discuté le choix du logiciel CLOVIS dans le chapitre I. Nous avions d'autre part la contrainte d'utiliser le VAX 780, contrainte fort agréable, puisque ce matériel nous a donné une entière satisfaction tant par sa fiabilité que par sa puissance.

L'équipe de recherche possède et utilise un TEKTRONIX 4114. Si cet outil est irréprochable pour sa fiabilité et sa haute résolution (grand nombre de points adressables sur l'écran), nous avons par contre beaucoup regretté le fait qu'il soit monochrome. L'usage de la couleur nous aurait permis de tester un plus grand nombre de concepts (notamment ceux exposés au chapitre III). Le fait que le poste de travail ainsi constitué ne comporte qu'un seul écran est également un handicap.

Enfin, le choix du langage PASCAL nous a semblé judicieux à cause de sa forte notion de type (Wir74). La manipulation de listes a été relativement aisée (nous ne disposions pas du langage LISP sous notre VAX) et a permis une mise au point rapide du programme.

Dépendance par rapport à ces choix

Nous nous proposons d'étudier jusqu'où va la dépendance de l'outil par rapport à ces choix.

a) CLOVIS

Le logiciel CLOVIS est indispensable pour faire fonctionner l'outil tel qu'il est actuellement. Cependant, il n'est réellement utilisé que dans un certain nombre de modules du programme.

La structuration a été faite de telle sorte que les parties graphiques soient nettement séparées des autres. Ainsi, en cas de changement de logiciel de base, seuls des modules qui font appel à CLOVIS devront être réécrits.

b) Le VAX et le langage PASCAL

Ici, il n'y a aucun problème de dépendance par rapport au matériel VAX mais le PASCAL sous système VMS n'est pas forcément le même que sous d'autres systèmes. D'après nous, les changements possibles sont dûs à la compilation séparée et sont donc assez faibles. En effet, nous n'avons pas utilisé de facilités de PASCAL qui ne sont pas "standard" (clause "else" dans le "case"...).

D'autre part, le VAX étant une machine avec pagination, il est possible de mettre en oeuvre de très gros logiciels. Le logiciel pour EDICAS est assez volumineux (environ 2500 lignes de PASCAL) d'autant plus qu'il nécessite le chargement du logiciel de base CLOVIS. Cependant, il faut être conscient du fait que l'utilisateur d'EDICAS utilisera également le système CASCADE. Il paraît donc difficile actuellement de le faire fonctionner sur un ordinateur "moyen" sans pagination du moins dans sa version actuelle.

c) Le TEKTRONIX

Aucune dépendance de l'outil graphique n'existe vis à vis du matériel graphique utilisé. C'est CLOVIS qui prend à sa charge cette indépendance. Nous remarquons d'ailleurs que des essais sont actuellement en cours pour faire marcher ce prototype sur un terminal couleur SONY de technologie totalement différente. (Bou84)

Nous conclurons en précisant que cette réalisation nous a permis de valider la structure graphique interne et qu'elle fera partie du prototype du système CASCADE du mois d'octobre 1984.

CHAPITRE III

REPRESENTATION GRAPHIQUE DU COMPORTEMENT

CHAPITRE III

REPRESENTATION GRAPHIQUE DU COMPORTEMENT

III.1 PARTIE COMPORTEMENTALE D'UN OBJET DECRIT EN CASCADE

Un module d'un système modélisé en langage CASCADE contient, en général, (Bre83), (Equ82), les parties suivantes:

- l'en-tête qui contient le nom du module, et la déclaration des attributs et de l'interface;
- les assertions qui précisent les contraintes sur les éléments d'interface, sur les attributs, et les relations existant entre les entrées et les sorties du modèle;
- le corps qui contient la partie de définition des "externes", la partie de déclaration d'objets locaux ou d'exemplaires de descriptions externes et un ensemble d'instructions décrivant le fonctionnement du module.

Toute la partie comportementale, c'est-à-dire les assertions et la partie fonctionnelle du corps, reste à décrire graphiquement. Nous exposons dans ce chapitre comment entrer graphiquement le comportement d'une description lorsque cela est possible. Cela revient à compléter EDICAS en permettant l'entrée graphique de descriptions comportant un fonctionnement précis à décrire. Nous préciserons comment le faire pour les assertions dans un premier temps, puis pour la partie fonctionnelle du corps.

III.2 EXPRESSION DES ASSERTIONS

L'un des objectifs du système CASCADE est de simuler les descriptions décrites avec le langage.

Les assertions permettent de spécifier le fonctionnement de la description. Elles sont constituées d'un ensemble de prédicats. Lors de la simulation, chacun de ces prédicats sera évalué à chaque pas de calcul. Lorsque l'un d'entre eux n'est plus vérifié, la simulation s'arrête car la description ne correspond pas à ce que l'on attendait. (fonctionnement différent de la spécification). Les évaluations des prédicats sont effectuées à chaque pas de calcul mais il est possible de les effectuer seulement à des instants précis de la simulation

(fenêtres de validité du prédicat). Un langage textuel permettant la spécification de ces assertions et de leurs fenêtres de validité a été entièrement défini: TPDL (BMB83). Parallèlement, un modèle mathématique pour l'observation temporelle des systèmes a été mis en place (Dec84). Les objets que l'on peut décrire avec TPDL sont définis en tant qu'objets mathématiques. L'approche choisie rejoint, en de nombreux points, l'approche de la logique temporelle.

Nous précisons ici, de manière informelle, comment on peut, à partir de ce langage définir les assertions et leur fenêtre ainsi que l'équivalence graphique du langage TPDL.

III.2.1 REPRESENTATION DES FENETRES DE VALIDITE

Les fenêtres de validité que l'on peut définir sont représentées comme un intervalle ou une union d'intervalles inclus dans $[0, +\infty[$ (demi droite des temps à partir de l'instant 0, temps de début de simulation).

Ces fenêtres peuvent être construites grâce à des fonctions de base auxquelles sont associés des intervalles de base d'une part et à des fonctions de composition qui permettent d'obtenir des intervalles qui ne sont pas des intervalles de base d'autre part.

Ainsi, chaque fenêtre de validité peut être obtenue par l'application d'une séquence de fonctions.

Remarque:

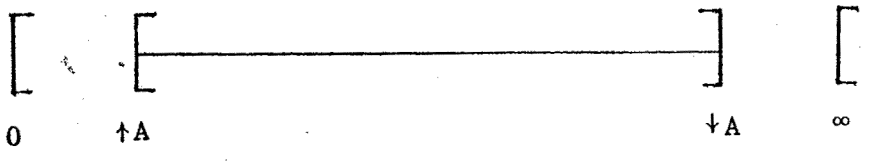
Dans tout ce qui suit, les bornes pourront être comprises ou non dans l'intervalle. La représentation exposée les comprend toujours.

a) Fonctions de base

a1. When A

"When" permet d'obtenir un ensemble d'intervalles qui décrit l'espace de temps pendant lequel une variable booléenne A est vraie.

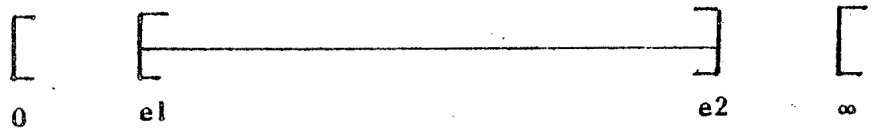
La représentation graphique sera la suivante:



a2. Within e1, e2

"Within" permet d'obtenir un ensemble d'intervalles qui décrit l'espace de temps compris entre l'arrivée d'un événement e1 et l'arrivée d'un événement e2.

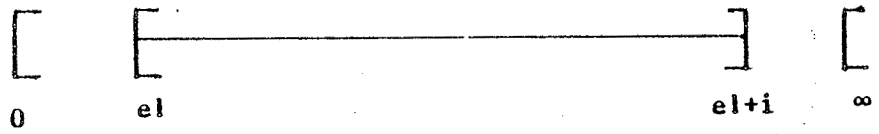
Représentation graphique de within:



a3. During e1, i

"During" permet d'obtenir un ensemble d'intervalles de temps entre l'arrivée d'un événement e1 et i unités de temps plus tard.

Représentation graphique:

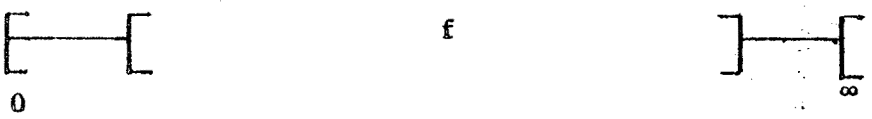


b) Fonctions de composition

b1. Except f

"Except" permet d'obtenir l'ensemble d'intervalles complémentaire à f dans R^+.

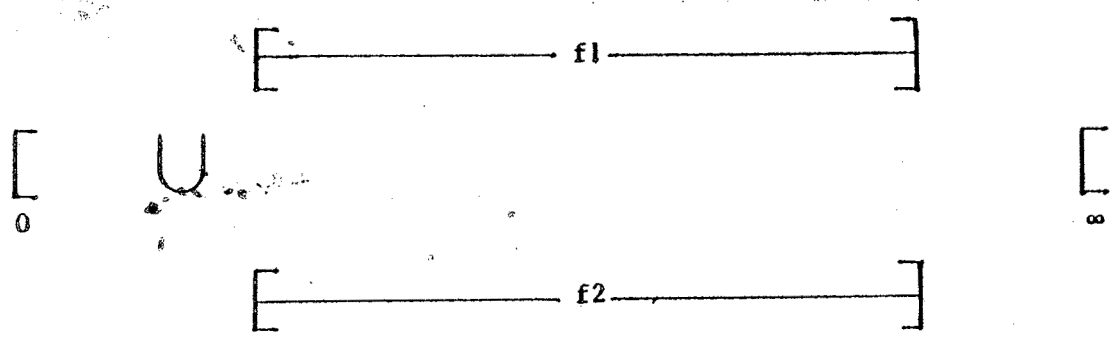
Représentation graphique:



b2. Union f1 f2

"Union" permet de faire l'union des deux ensembles d'intervalles f1 et f2.

Représentation graphique:



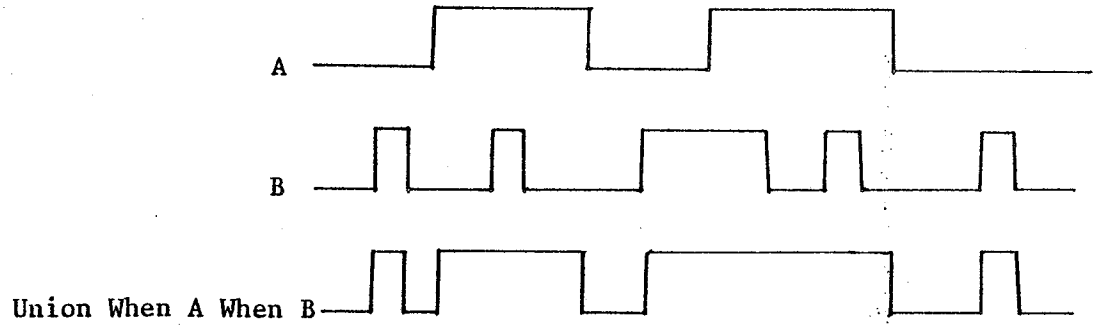
Remarque:

Cette représentation ne signifie pas que les fenêtres f1 et f2 sont alignées dans le temps. Il s'agit ici d'une représentation compactée.

Exemple: Soit l'événement Union when A when B.

Avec notre représentation, il est représenté par:

Sur une échelle des temps classiques:

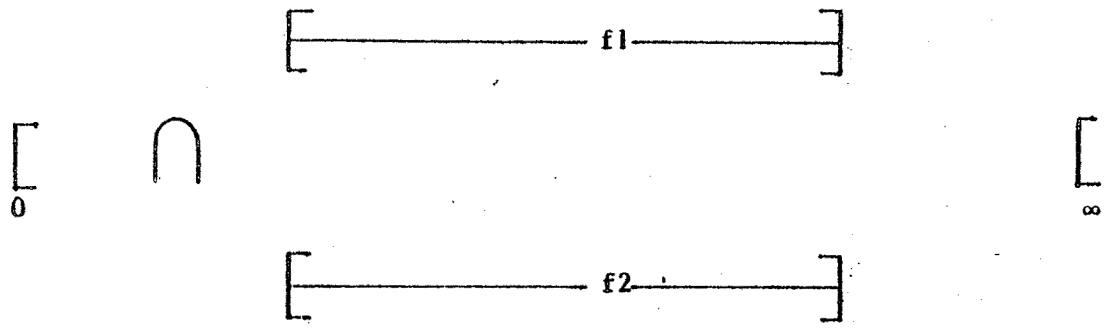


Cette représentation ne peut pas être choisie pour notre application puisque les valeurs de A et B ne sont connues qu'à la simulation.

b3. Intersection f1 f2

"Intersection" permet de faire l'intersection entre deux ensembles d'intervalles f1 et f2.

Représentation graphique:



Remarque: la même remarque que pour Union reste valable.

III.2.2 REPRESENTATION DES EVENEMENTS

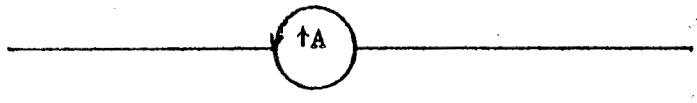
De la même manière que les fenêtres de validité, les événements sont construits à l'aide de fonctions de base et de fonctions de composition.

a) Fonctions de base

a1. Anytime A

"Anytime" permet de générer un événement lorsque la variable booléenne A passe de `faux à `vrai.

Représentation graphique:



Remarque: le symbole ◯ exprime la répétition possible de l'événement.

a2. Anychange A

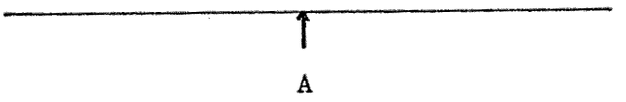
"Anychange" permet d'obtenir un événement lorsque la variable booléenne A est modifiée.

Représentation graphique:

a3. As_soon_as A

"As_soon_as" permet d'obtenir un événement dès que A est vrai.

Représentation graphique:

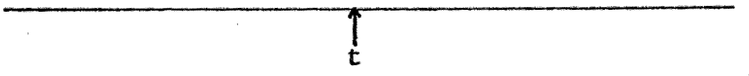


Remarque: le symbole ◯ n'est plus utilisé car l'événement n'est pas répétitif. Le symbole ↑ veut montrer l'unicité d'arrivée d'un tel événement (un seul point est montré sur la droite des temps).

a4. At t

"At" permet de définir un instant précis: t unités de temps après 0.

Représentation graphique:

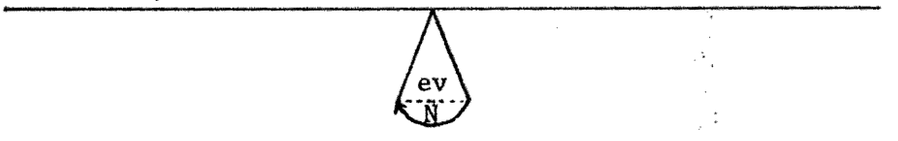


b) Fonctions de composition

b1. The time N that event

"The time... that... " permet à partir d'un événement répétitif d'obtenir un événement simple:

Représentation graphique:



Remarque: pour exprimer que l'on attend que la répétition de l'événement ev se produise N fois avant de générer un événement unique, le mélange des formes "cercle" et "triangle" semble un bon symbolisme: on ne montre qu'un point sur la droite des temps tout en gardant la notion de répétition grâce à l'arc de cercle.

b2. Every i starting from event

"Every... starting from... " permet d'obtenir un événement répétitif à partir d'un événement quelconque. La période de répétition est de i unités de temps.

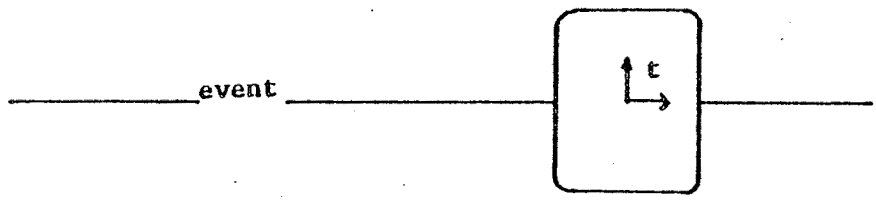
Représentation graphique:

Remarque: le symbole choisi ici est l'hexagone. Ce choix est justifié par le fait que l'hexagone rappelle la forme du cercle en marquant une certaine périodicité (angle).

b3. Delay t for event

"Delay... for... " permet de définir un événement à partir d'un autre en le retardant de t unités de temps.

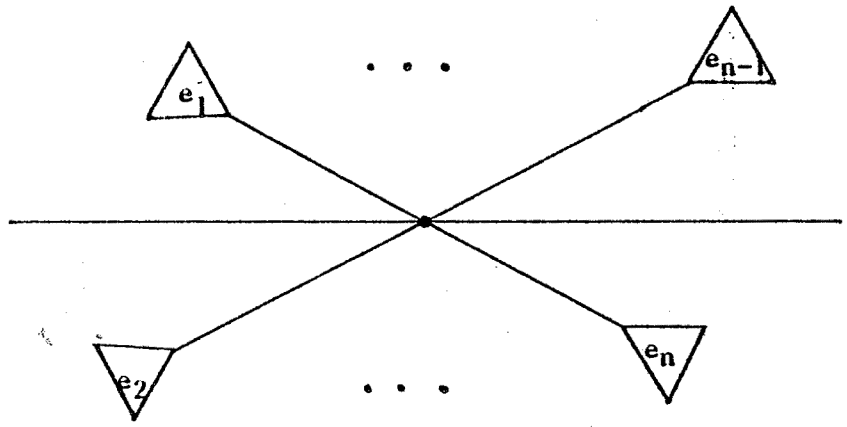
Représentation graphique:



b4. In any order e1 e2... en

"In any order" permet à partir de n événements d'en obtenir un nouveau lorsque ces n événements arrivent l'un après l'autre dans n'importe quel ordre.

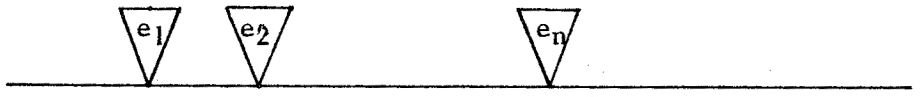
Représentation graphique



b5. In order e1 e2... en

"In order" permet de générer un événement lorsque les événements e1, e2, ..., en se sont tous produits et exactement dans cet ordre.

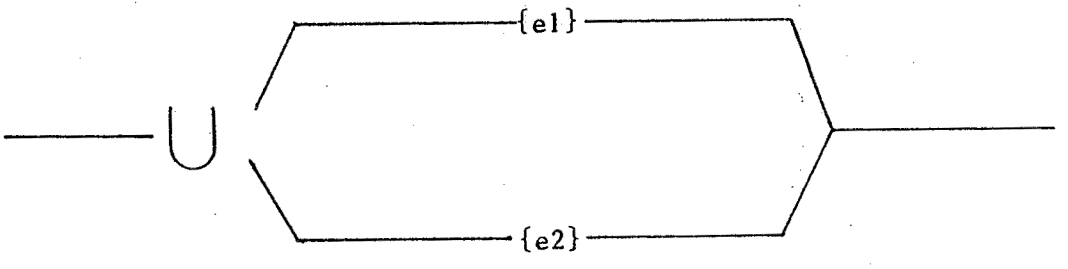
Représentation graphique:



b6. Union e1 e2

"Union" permet de générer un événement lorsque l'un des deux événements e1 ou e2 arrive.

Représentation graphique



III.2.3 REPRESENTATION DES LIENS ENTRE FENETRES DE VALIDITE ET EVENEMENTS

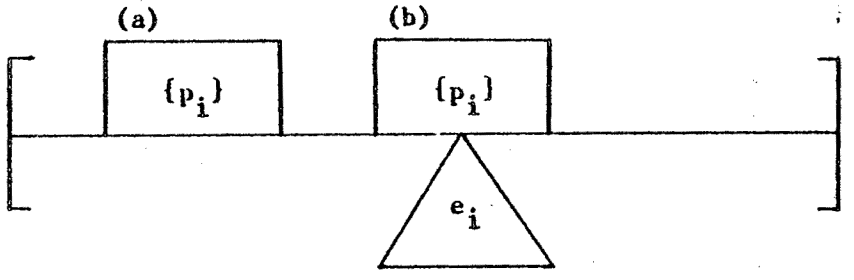
Lorsqu'il décrit ses spécifications, l'utilisateur décrit un ensemble de prédicats à vérifier dans certaines fenêtres de validité.

Dans chaque fenêtre de validité, on peut vouloir exprimer

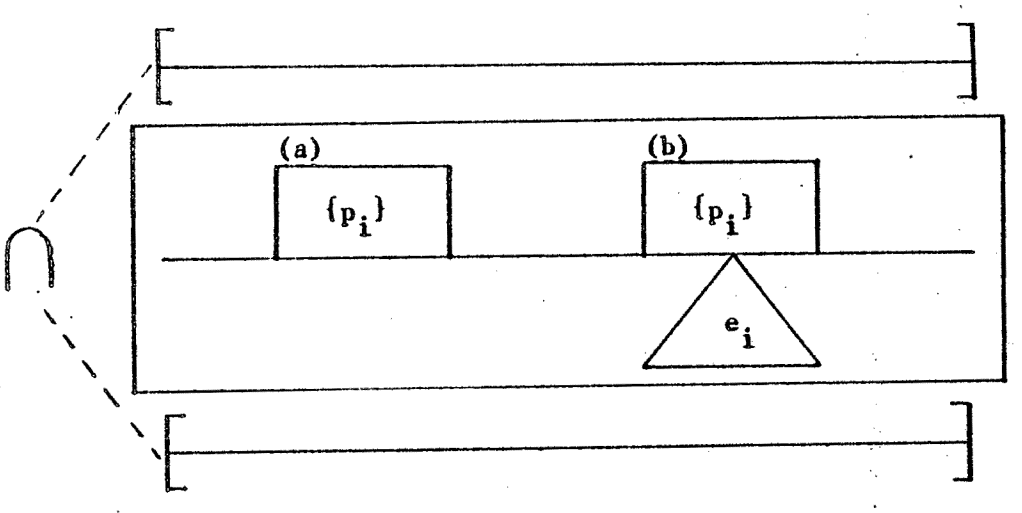
- soit qu'un ensemble de prédicats doit être toujours vrai dans cette fenêtre (fig a);
- soit qu'un ensemble de prédicats doit être vérifié lorsqu'un événement arrive (fig b).

Les symboles graphiques sont les suivants:

* sur les intervalles de base



* sur les intervalles composés (obtenus à partir des fonctions except, union, intersection)



Remarque: l'encadré peut être remplacé par "reverse vidéo" si le matériel le permet.

III.2.4 EXEMPLE: ALU 74LS181

Considérons l'unité arithmétique et logique 74LS181 définie dans le manuel (TI77) et dont l'interface de communication est la suivante:

en entrée:

- deux opérandes A et B représentés sur quatre bits;
- S: sélection de l'opération à effectuer, également sur quatre bits;

- M représenté sur un bit qui indique si le mode est arithmétique ou logique;
- CN, représenté sur un bit, valeur de la retenue entrante.

en sortie:

- F, résultat, représenté sur quatre bits;
- un bit CN_PLUS_4, valeur de la retenue sortante, en mode arithmétique;
- G et P, bits de génération et de propagation de retenue, en mode arithmétique;
- un bit A_EQ_B qui indique l'égalité des deux opérandes A et B.

D'autre part, F devra contenir la bonne valeur résultat, au maximum 38 unités de temps en mode logique et 32 unités de temps en mode arithmétique (26 unités de temps pour CN).

Ces spécifications se traduisent immédiatement en CASCADE par la description suivante (extraite de (BMB83)).

```
reflan CASSANDRE_AS
description ALU (in TERMINAL (BOOL,0) A[0:3], B[0:3], S[0:3], M, CN;
                 out TERMINAL (BOOL, 0) F[0:3], A_EQ_B, G, P, CN_PLUS_4)

assert
     when M then
          delay 38 for (anytime(%1!A~=A) union anytime(%1B~=B))
              then F = EVALLOGIC(%38!A, %38!B, %38!S)
          end
     else
          delay 26 for anychange CN union
          delay 32 for (anytime(%1!A~=A) union anytime(%1!B~=B))
              then F = EVALARITH(%32!A, %32!B, %32!S, %26!CN)
          end
end
     ...
corps
     ...
fin description.
```

où . EVALARITH et EVALLOGIC sont des fonctions déjà définies en CASCADE qui permettent de calculer F en tenant compte des différentes entrées.

. ~= est le symbole "différent"

et . %i!A donne la valeur de A i unités de temps avant.

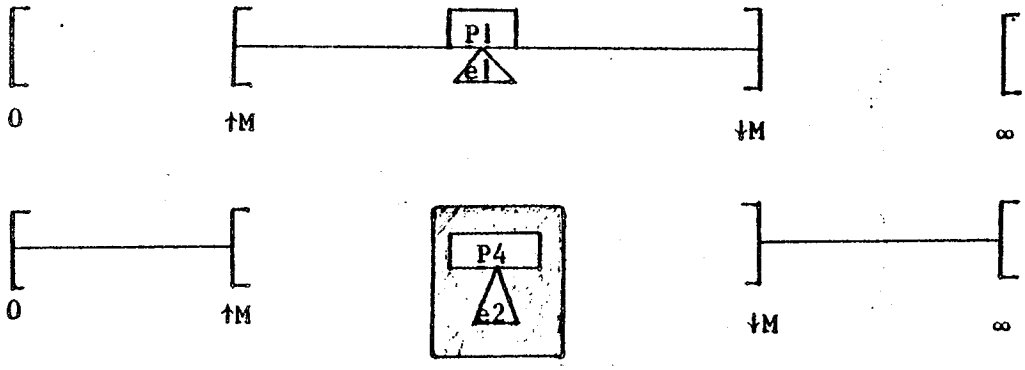
Remarquons que anytime(%1!A~=A) génère un événement dès qu'un bit de A est différent par rapport au top précédent.

Equivalence graphique de l'assertion ainsi représentée

Posons pour simplifier le graphisme

P1 = (F = EVALLOGIC(%38!A, %38!B, %38!S))
P2 = (%1!A~ = A)
P3 = (%1!B~ = B)
P4 = (F = EVALARITH(%32!A, %32!B, %32!S, %26!CN))

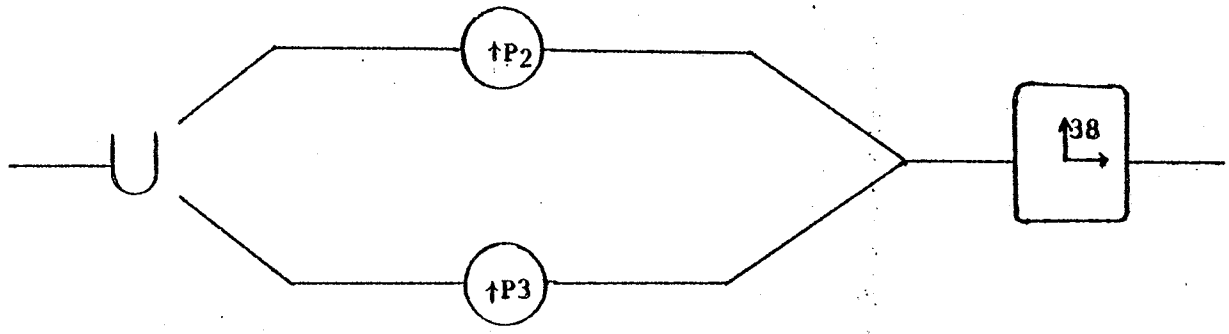
Le schéma obtenu est le suivant :



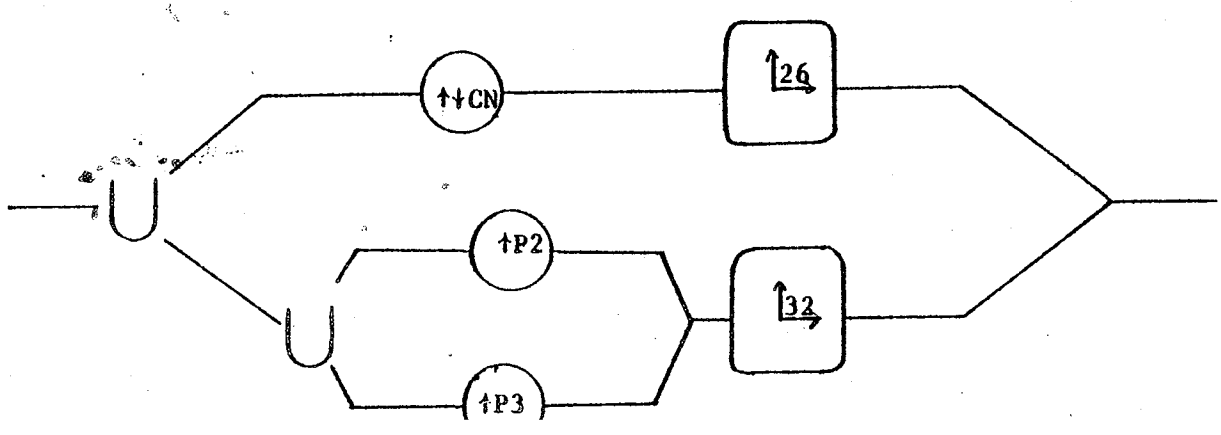
où e1 et e2 sont décrits ci-dessous.

Les fenêtres de validité associées aux prédicats P1 et P4 à vérifier sont complémentaires (ce qui explique l'effet "inverse-vidéo" pour la deuxième ligne).

e1≡



e2≡



III.2.5 CONCLUSION SUR LA REPRESENTATION GRAPHIQUE DES ASSERTIONS

Nous prévoyons en extension de l'éditeur graphique actuel, une entrée possible des assertions sous forme graphique.

L'équivalence entre les assertions textuelles et graphiques étant parfaitement définie, les traductions dans les deux sens sont possibles (assertion graphique ----> assertion textuelle et assertion textuelle ----> assertion graphique).

Nous avons donc ainsi étendu les deux fonctions $\emptyset$ et $\emptyset^{-1}$ définies dans le chapitre II de manière à ce que la traduction prenne également en compte une partie comportementale.

III.3 REPRESENTATION GRAPHIQUE DU COMPORTEMENT

La partie fonctionnelle d'une description est décrite différemment suivant le niveau de langage auquel on se trouve. De nombreuses notions sont communes pour la description de ces parties fonctionnelles mais certaines sont spécifiques au niveau de langage considéré.

Nous présentons dans un premier temps les notions communes aux différents niveaux de langage et leur représentation graphique lorsqu'elle existe. Puis nous étudions les parties spécifiques de deux niveaux de langage très importants en proposant un graphisme associé:

- les niveaux LASCAR et CASSANDRE que l'on peut souvent regrouper à cause de leur grande similitude. Ils permettent de décrire des objets au niveau transfert de registre (CASSANDRE) et au niveau architecture et spécification des composants (LASCAR);
- le niveau LASSO qui permet la modélisation au niveau "architecture de systèmes informatiques".

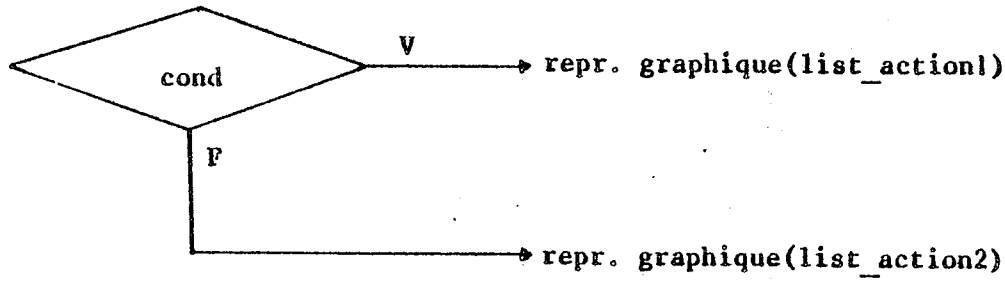
III.3.1 PRESENTATION DES NOTIONS COMMUNES AUX DIFFERENTS NIVEAUX DE LANGAGE

a) La forme conditionnelle "si"

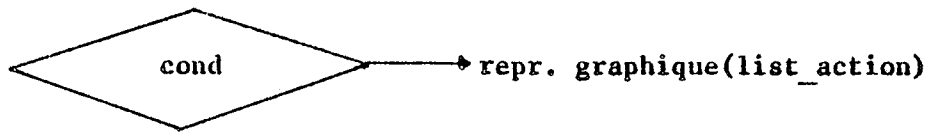
Cette forme permet d'exprimer une alternative de la manière habituelle: si cond alors list_action1 sinon list_action2 finsi où cond est une expression booléenne, list_action1 et list_action2 des ensembles d'instructions de la partie relations.

Il existe déjà une représentation graphique de cette forme pour les organigrammes. Nous l'avons adoptée:

représentation graphique:



La partie "sinon" étant optionnelle, la représentation graphique devient:



b) La forme conditionnelle "cas"

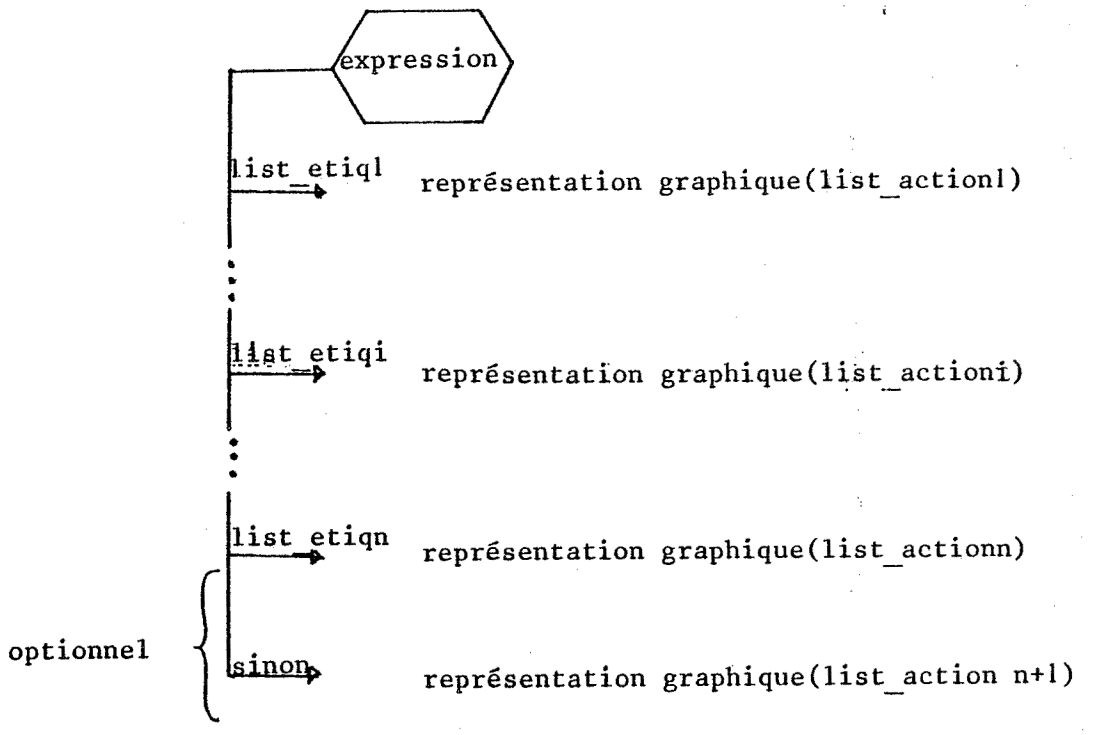
Là encore, il s'agit d'une forme habituelle qui permet de sélectionner une action parmi n suivant la valeur d'une expression:

```
cas expression est

    liste_etiq1 :  liste_action1;
    liste_etiq2 :  liste_action2;
                   .
                   .
                   .
  liste_etiq n :  liste_action n
       sinon liste_action n+1 fincas
```

où "expression" a pour résultat une valeur entière littérale, logique;
."liste_action i" sont des ensembles d'instructions de la partie relations.

représentation graphique:



c) Les connexions

Nous avons déjà vu comment connecter des éléments de manière permanente (utilisation de l'instruction "unités") grâce à l'interface effective des éléments. Il est cependant nécessaire de décrire des connexions qui ne s'effectuent que sous certaines conditions. Ceci se fait dans la partie de description fonctionnelle puisque ces connexions ne sont pas seulement structurelles.

Représentation graphique:

Nous voyons ici apparaître une intersection entre parties fonctionnelle et structurelle. Nous tracerons donc les connexions entre les éléments dans la partie structurelle de la même manière que pour les connexions permanentes mais en mettant les traits en pointillé pour montrer que cette connexion n'est pas forcément valide. Nous donnerons un nom à cette connexion auquel nous ferons référence dans la partie graphique associée au comportement.

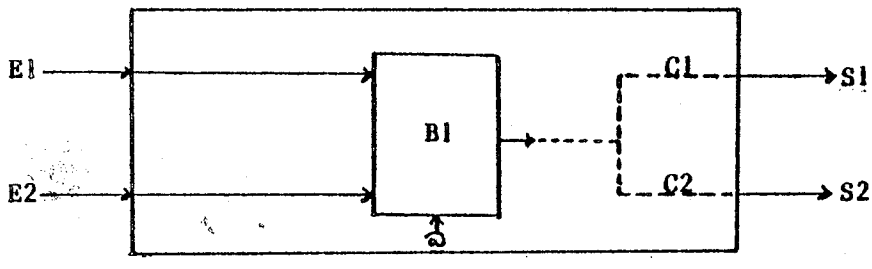
Exemple: Soit la description A utilisant un exemplaire de la description B:B1.

```
description A(in SIGBO E1,E2;  out  SIGBO S1,S2);
     .
     .
     .
déclare SIGBO COM;
unités B B1(E1, E2,,);
     .
     .
     .
relations
     .
     .
     .
B(,, com,);
     .
     .
     .
si COND alors B(,,, S1) sinon B(,,, S2);
     .
     .
     .
fin

avec description B(in SIGBO A, B, C;  out  SIGBO D);
```

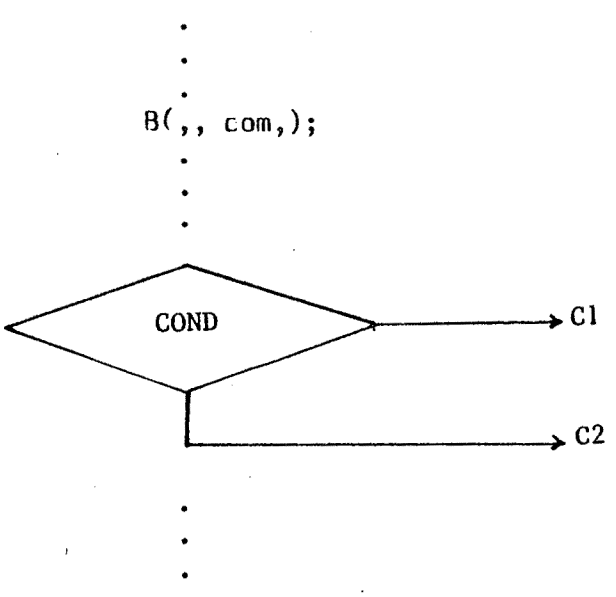
L'équivalent graphique est:

structurel:



ⓐ indique la liaison à une expression d'un élément d'interface

fonctionnel:



Remarques:

- Des connexions permanentes sont établies entre E1 et B1.A et entre E2 et B1.B.
- Une connexion de l'élément B1.C à l'expression "com" locale à la description A est effectuée. Ceci explique le symbole spécial utilisé sur le schéma structurel.
- Une connexion conditionnelle de B1.D avec S1 ou S2 est représentée en pointillé.

d) Les expressions

Les expressions en CASCADE sont constituées, comme dans de nombreux langages, d'opérateurs et d'opérandes. Nous n'entrerons pas en détail dans la syntaxe des expressions, celle-ci étant décrite dans (Equ82) et dans (Bre83).

C'est pour cette partie que l'éditeur graphique de CASCADE mérite le plus le nom d'éditeur mixte graphique/textuel. En effet, bien qu'il soit possible d'associer à chaque opérateur un graphisme particulier, nous ne voyons guère l'intérêt d'une telle transformation. Cela obligerait en effet l'utilisateur à connaître toutes les correspondances opérateurs/graphismes sans apporter plus de clarté. Nous laisserons donc les expressions sous leur forme textuelle.

III.3.2 LES NIVEAUX LASCAR ET CASSANDRE (BRE84A)

Les niveaux LASCAR et CASSANDRE du langage CASCADE permettent de décrire des objets à des niveaux assez "hauts" ("transfert de registres" et "architecture et spécification des composants"). La partie contrôle d'un circuit séquentiel est modélisée par un automate d'états finis déterministe.

Avec le langage CASCADE, un automate se présente comme une suite d'états sous forme de blocs étiquetés. Une instruction spéciale "charger" permet de changer d'état dans l'automate.

Exemple: Voici l'automate décrivant un multiplieur 8 bits: (Lef84)

```
"reflan LASCAR_S description AUTOMATE
          (in HORLOGE H;  in SIGBO EM[0:7], DEPART,
           S[0:7], RS;  out REGBO A[0:7], B[0:7];
           out SIGBO SM[0:15], OCCUPE)
corps
declare SIGBO RC0, RC1, RC2;
        REGBO BUSY, C[0:2], A1[8:15], R;
relation
    OCCUPE .= BUSY, RC2 .= `1,
                    RC1 .= RC2 & C[2],
                    RC0 .= RC1 & C[1],
    :REPOS:       !H!  B <= EM, si DEPART alors charger INIT,
                                                BUSY <= 0 fin;
    :INIT:        !H!  A <= `00000000, A1 <= B,
                       B <= EM, C <= `000,
                       charger ADDITION, BUSY <= 1;
    :ADDITION:    si A1[15] alors !H!  A <= S, R <= RS;
                              sinon !H!  R <= 0;  fin;
                  !H!  C <= (RC0\RC1\RC2) xor C, charger DECAL;
    :DECAL:       !H!  A <= R\A[0:6],
                       A1 <= A[7]\A1[8:14];
                  si reduc|C alors !H!  charger ADDITION;
                              sinon !H!  charger FIN;
    :FIN:         SM .= A\A1,
                  !H!  charger REPOS;
fin"
```

Représentation graphique

Dans la description CASCADE associée à un automate, nous décrivons simultanément deux parties: la partie contrôle qui décrit le séquencement de l'automate et la partie opératoire qui décrit les actions associées à chaque état de l'automate.

partie contrôle:

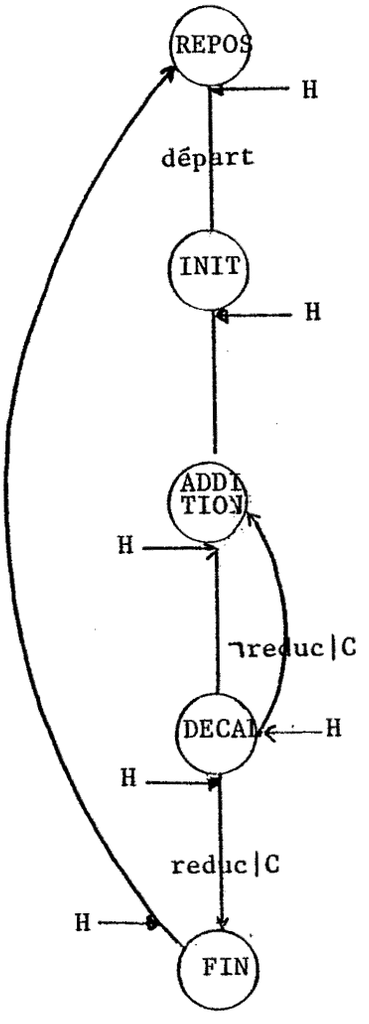
Un automate est défini par un ensemble d'états et un ensemble de transitions entre ces états. Nous pouvons donc représenter le séquencement de l'automate par un ensemble de cercles (les états) reliés entre eux par des arcs (les transitions). Ces arcs peuvent éventuellement être associés à une condition (condition nécessaire pour que la transition ait lieu) et à une condition d'horloge (top

d'horloge sur lequel on validera l'état suivant).

Exemple: reprenons l'exemple précédent en n'exprimant que le séquencement de l'automate.

```
:REPOS:    !H!  si départ alors charger INIT fin;
:INIT:     !H!  charger ADDITION;
:ADDITION: !H!  charger DECAL;
:DECAL:    !H!  si reduc|C alors charger ADDITION
                        sinon charger FIN;  fin;
:FIN:      !H!  charger REPOS;
```

Ce séquencement sera représenté graphiquement par:



Ce schéma doit être interprété selon les conventions habituelles de représentations graphiques des automates:

* Un jeton se "promène" dans le graphe. Initialement, il est dans la case "REPOS" (ceci est arbitraire).

* A chaque top de l'horloge H, les conditions associées aux transitions qui partent de l'état courant (état où se trouve le jeton) sont évaluées. Si rien n'est mentionné sur une transition, on lui associe la condition "vrai".

* Si aucune des conditions n'est vérifiée, on reste dans le même état et on attend le top d'horloge suivant.

* Si une condition est vérifiée, on fait passer le jeton dans le cercle représentant le nouvel état validé.

* A tout instant, un seul état est validé.

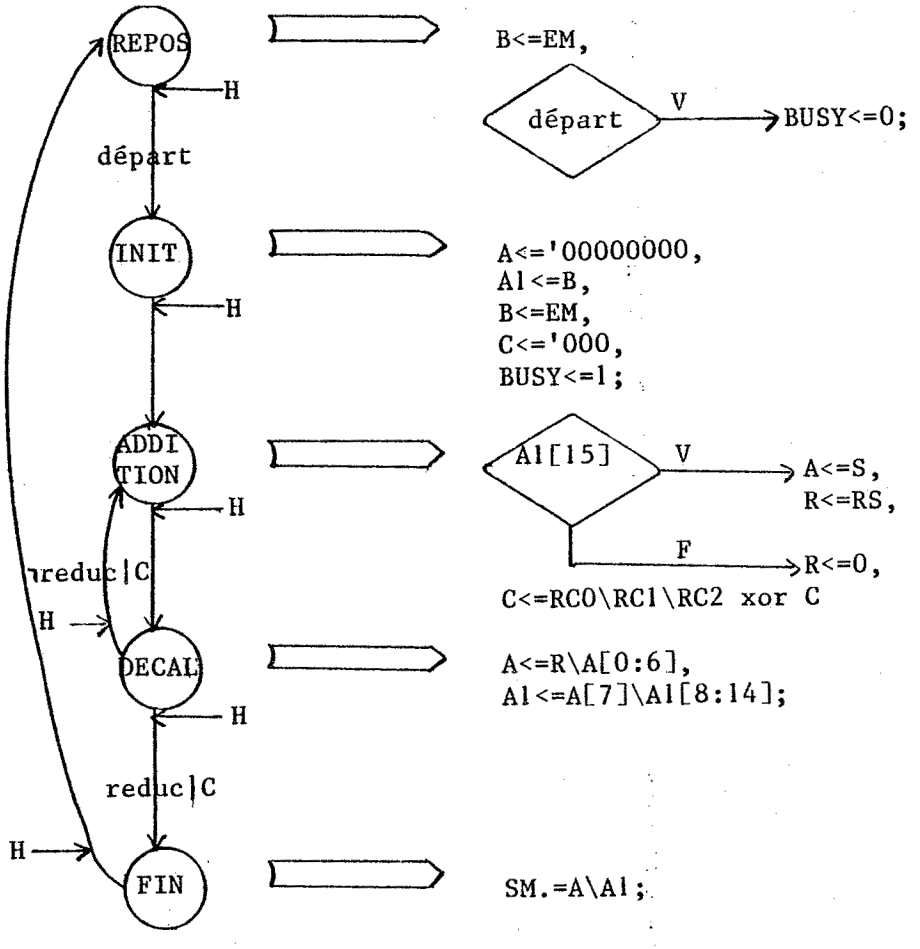
* L'intersection des conditions associées aux transitions partant d'un état est vide. En effet, si deux conditions étaient vraies en même temps, cela permettrait le non déterminisme que nous ne voulons pas exprimer ici.

* Lorsqu'il y a une horloge et une seule, les H--> sont générés automatiquement.

partie opératoire

A chaque état est associé un ensemble d'actions à effectuer. Ces actions sont du type de celles décrites dans la partie des notions communes (III.3.1). Nous connaissons donc le graphisme qui leur est associé.

Exemple: dans le cas de l'automate du multiplieur 8 bits, la représentation graphique est la suivante:



III.3.3 LE NIVEAU LASSO (BRE84B), (BOR81)

Nous pouvons grâce au niveau LASSO décrire les architectures de systèmes informatiques. A ce niveau de langage, c'est l'aspect fonctionnel des composants qui nous intéresse. La principale caractéristique de ce niveau de langage est la description du graphe de contrôle des objets à modéliser. Le graphe de contrôle est constitué d'un ensemble de transitions. Ces transitions sont de la forme: ?condition?transition.

La condition est un signal, ou le ET de plusieurs signaux de contrôle. Elle permet, si elle est vraie, d'entrer dans la transition. La partie transition proprement dite se compose:

- d'une partie opérative optionnelle (début... fin) qui s'exécute lorsqu'on entre dans la transition;

- de l'expression de durée de la transition optionnelle (délai). Si cette partie est omise, la transition est instantanée;

- du type de la transition optionnel (primitive). Ceci précise la nature de la transition. Le détail de toutes ces primitives est donné dans (BoG80). Nous trouverons ci-dessous une explication de ces primitives et leur représentation graphique. Dans le cas où la primitive est omise, on considère qu'il s'agit d'une transition au sens des réseaux de Pétri (Sif79). La représentation graphique sera alors un simple rectangle avec éventuellement le délai à l'intérieur si la transition n'est pas immédiate;

- de la liste des signaux à valider (valider) après la transition.

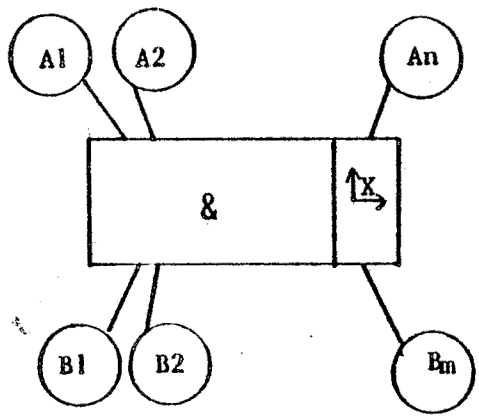
Donnons un aperçu des types de transition que l'on peut rencontrer dans le graphe de contrôle:

a) Expression de l'attente d'arrivée d'un ou plusieurs signaux

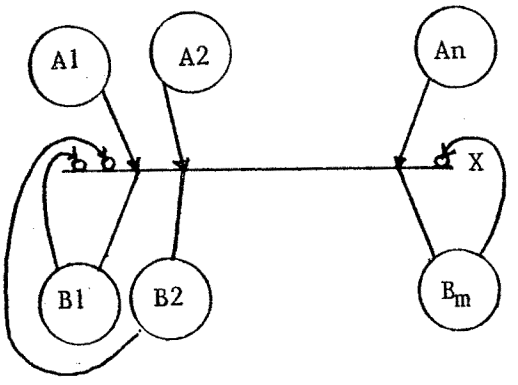
?A1&A2&... &An? délai(X) valider (B1, ..., Bm)

Les Ai sont les signaux en entrée de la transition. Les Bi sont en sortie. Lorsque tous les Ai sont à 1 et tous les Bi à 0, la transition est effectuée (tous les Bi sont mis à 1 et les Ai à 0 après X unités de temps).

Représentation graphique



Remarque: cela peut correspondre à la représentation suivante avec les réseaux de Pétri:



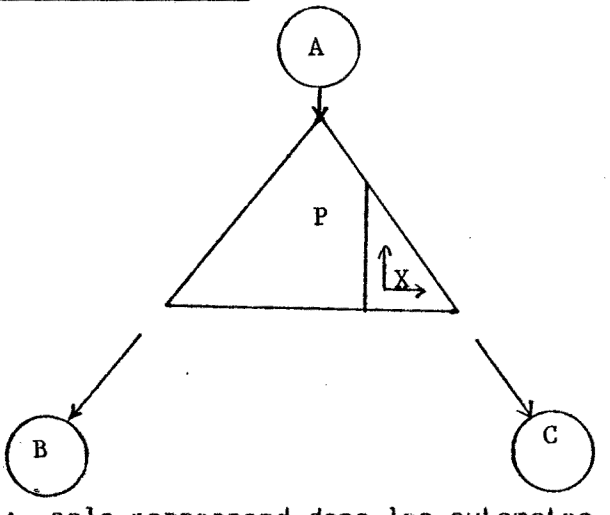
Il s'agit de réseaux de Pétri temporisés où X est le délai à attendre en cas de possibilité d'activation de la transition et où ----○ teste si les places n'ont pas de marques (ou de jetons).

b) Expression d'un choix après l'arrivée d'un signal

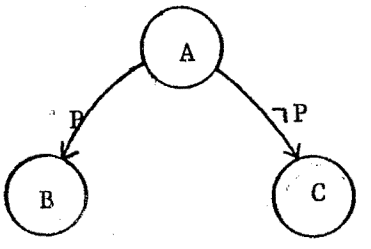
?A? selon P délai (X) valider (B, C)

A est un signal d'entrée de la transition. B et C sont des signaux de sortie. Lorsque les signaux de sortie sont à 0 et le signal d'entrée est à 1, on évalue le prédicat P. S'il est vrai, B est validé, sinon C. A est remis à 0. X est le délai de l'opération.

Représentation graphique

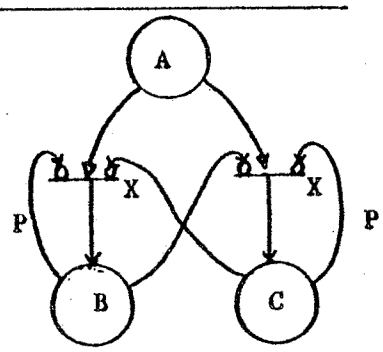


Remarque: cela correspond dans les automates à :



mais sans tenir compte de la contrainte de temps: délai (X)

Représentation avec réseaux de Pétri

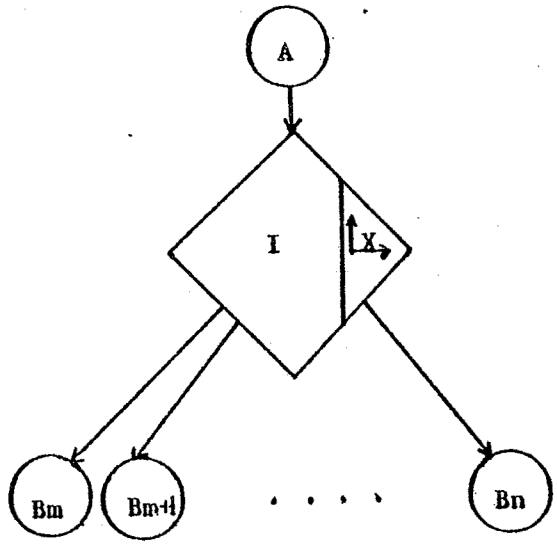


c) Expression d'un choix entre plusieurs actions après l'arrivée d'un signal

?A? index I de m à n délai (X) valider (Bm, Bm+1, ...,Bn)

Cela se passe de la même façon que pour le choix entre deux actions mais au lieu d'évaluer un prédicat, on évalue un entier qui validera un des signaux.

Représentation graphique

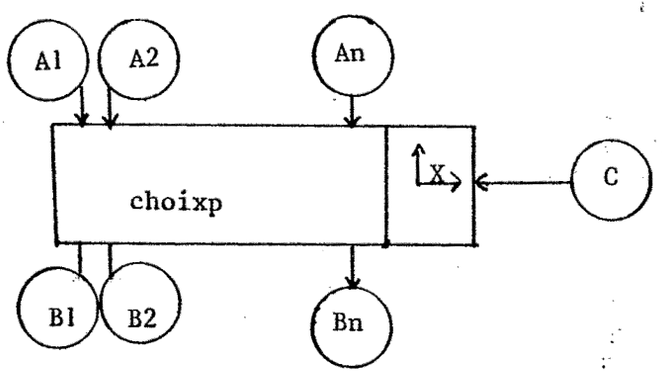


d) Expression d'un choix avec priorité

?C? choixp(A1, ..., Am) délai (X) valider (B1, ...,Bn)

C et les Ai sont les signaux d'entrée de la transition. Les Bi, en bijection avec les Ai, sont les signaux de sortie de la transition. Si tous les Bi sont à 0 et C est à 1, on regarde parmi les Ai qui sont à 1 le plus prioritaire (priorité de gauche à droite) et on valide le Bi correspondant; on remet le Ai à 0.

Représentation graphique

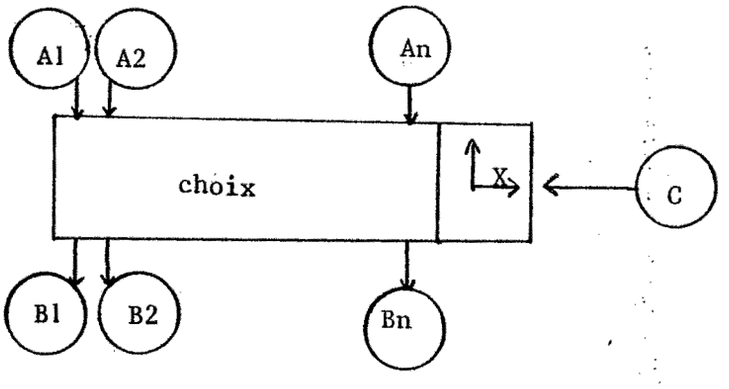


e) Expression d'un choix d'une action et une seule avec priorité égale

?C? choix (A1, ..., An) délai (X) valider (B1, ..., Bn)

Il se passe la même chose qu'avec priorité mais ici une seule place Ai doit être valide car la priorité est égale pour tous les Ai. Dans le cas contraire, il y a erreur.

Représentation graphique



(L'erreur pourra se traduire à la simulation par un clignotement).

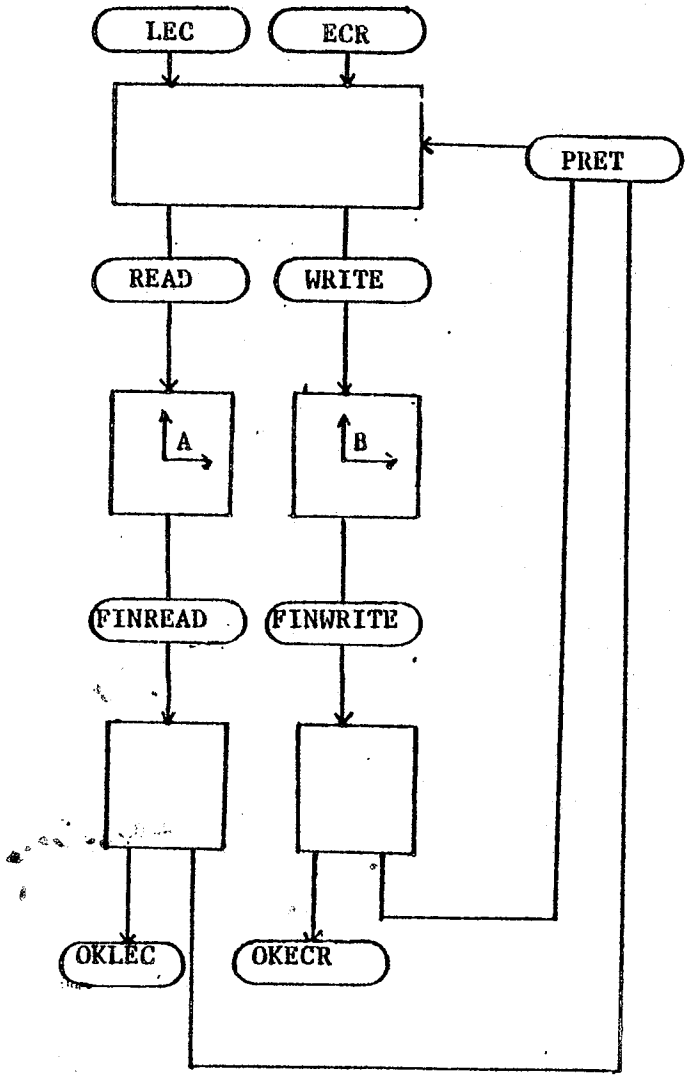
f) Exemple

Description d'une mémoire dont la taille est N, le temps de lecture est A et le temps d'écriture est B.

Le texte CASCADE est :

```
reflan LASSO description MEMOIRE (ENT N, A, B)
             (in CONTROLE LEC, ECR;  in ENT AD, DATAE;
              out CONTROLE OKLEC, OKECR;  out ENT DATAL)
corps
   declare ENT TABLE[1:N];
           SIGBI PRET;
relations
   ?PRET?     choix (LEC, ECR) valider (READ, WRITE);
   ?READ?     début DATAL:=TABLE[AD];
              fin délai (A) valider (FINREAD);
   ?WRITE?    début TABLE[AD]:=DATAE;
              fin délai (B) valider (FINECR);
   ?FINREAD?  valider (OKLEC, PRET);
   ?FINECR?   valider (OKECR, PRET);
 .
 .
 .
```

Le graphe de contrôle de la partie relations sera entré graphiquement par :



Remarque: l'association avec la partie textuelle de la description est faite de la même manière que pour le niveau LASCAR.

III.3.4 CONCLUSION SUR LA REPRESENTATION GRAPHIQUE DU COMPORTEMENT

Cette partie termine le développement des méthodes de traduction d'une description textuelle CASCADE en une description graphique (et réciproquement).

Nous avons donc défini comment entrer avec l'éditeur graphique de CASCADE toute description que l'on pouvait décrire textuellement. De même, nous avons défini comment trouver, à partir d'une description textuelle, son équivalent graphique. Ceci a constitué l'essentiel de notre travail.

III.4 LA COULEUR DANS L'EDITEUR GRAPHIQUE DE CASCADE

Il est possible d'étendre les possibilités de l'éditeur graphique décrites précédemment grâce à l'utilisation de la couleur. C'est cette extension que nous exposons dans ce paragraphe.

III.4.1 RAPPEL SUR LA REPRESENTATION DES COULEURS

Pour définir une couleur, plusieurs systèmes de représentation sont possibles (systèmes RGB, HSV, IQY, dictionnaires de couleur, ...) (BGK82), (JoG78). Le système HSV est le plus courant. L'espace des couleurs est représenté par un double cône (fig 1). Une couleur est donnée par ses coordonnées cylindriques. L'angle θ (dans le plan Oxy) représente la couleur dans une palette prédéfinie (fig 1). La coordonnée radiale (module) comprise entre 0 et 1 représente la saturation: si cette coordonnée est proche de 0, la couleur sera dans les tons de gris; si elle est proche de 1, la couleur sera très vive. La coordonnée sur l'axe Oz (côte), comprise entre 0 et 1, représente la brillance de la couleur (noir si proche de 0, blanc si proche de 1).

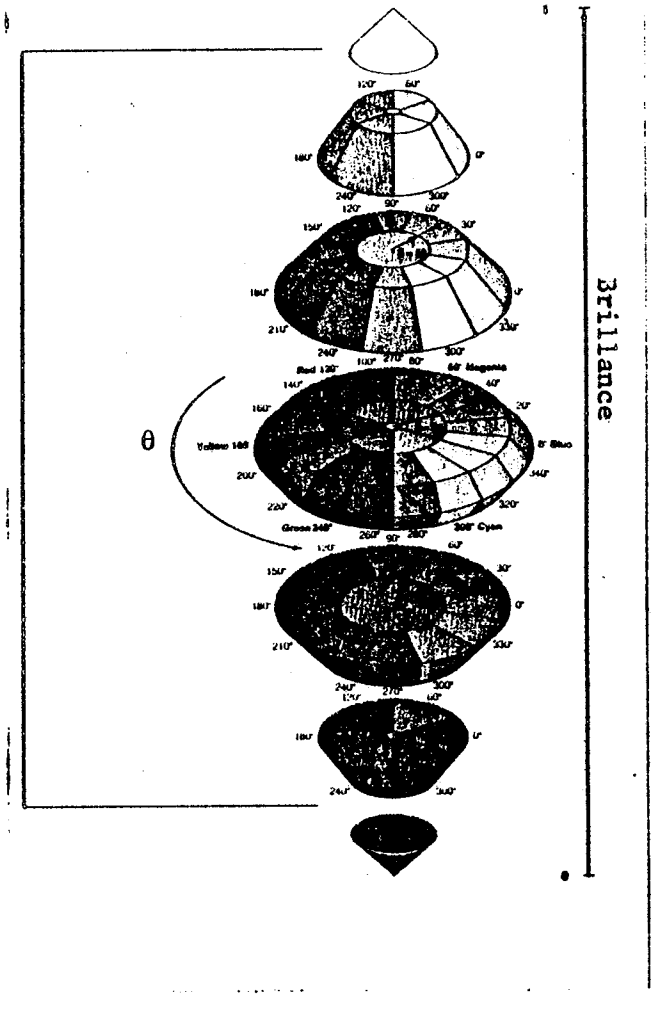


fig 1: double cône

III.4.2 UTILISATION DE LA COULEUR DANS L'EDITEUR GRAPHIQUE

a) Généralités

La couleur est utilisée en général dans deux cas bien distincts:

- elle permet de clarifier un dessin sans pour autant avoir une signification par rapport aux objets du dessin;
- elle permet de définir une relation d'équivalence entre objets: lorsque l'on regarde deux objets de la même couleur, on est tenté de les rapprocher. La couleur fournit alors une information supplémentaire au dessin de base.

Exemples d'utilisation possible:

- des boîtes d'une même couleur sont définies au même niveau de langage;
- des fils de connexions de même couleur prouvent que le même type d'informations circule sur ces différents fils...

D'autre part, une couleur peut s'exprimer à l'aide de trois paramètres. On peut associer à chacun d'eux une information. Ainsi, au lieu de rapprocher seulement les objets ayant la même couleur, on peut rapprocher les objets proches d'une certaine couleur (θ à peu près égal), les objets très saturés (vifs), ou bien les objets très brillants).

Exemple: plus l'angle de la couleur représentant une boîte est grand, plus le niveau de langage est élevé (Bleu pour LASSO (système) θ=320°, Jaune pour CASSANDRE (logique) θ=180°, Rouge pour IMAG (électrique) θ=80°).

b) Contour et intérieur

On peut associer deux couleurs à certains objets. En effet, les objets composés d'un contour fermé et d'une surface intérieure à ce contour peuvent être représentés de manière bicolore (une couleur pour le contour et une pour l'intérieur).

Exemple d'application: si une boîte est décrite dans un certain niveau de langage, la couleur du contour peut représenter ce niveau de langage tandis que l'intérieur de la boîte n'est colorié que s'il s'agit d'une boîte non décomposable (feuille de l'arbre des imbrications).

c) Notion de gomme

Lorsqu'il dessine sur une feuille de papier, l'utilisateur dispose de plusieurs matériaux ayant pour lui un sens:

- un crayon à papier (premier essai);
- un stylo encre et des couleurs (dessin plus sûr);
- l'encre de chine (dessin définitif).

Il peut facilement gommer le crayon à papier, à la limite le stylo et les couleurs mais il ne peut pas effacer l'encre de chine.

L'utilisateur peut exprimer ses doutes de cette manière en donnant une couleur très saturée s'il est sûr de ce qu'il dessine et très grise dans le cas d'une première ébauche.

d) Proposition d'utilisation de la couleur pour EDICAS

Dans tout ce paragraphe, nous raisonnons sur deux niveaux:

. un niveau riche (l'utilisateur dispose de la couleur);

. un niveau pauvre (il n'en dispose pas).

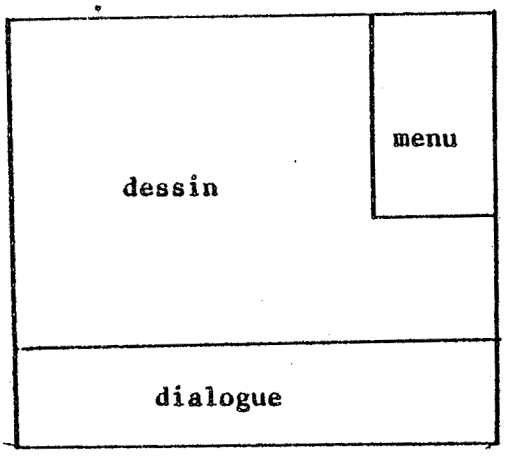
1) Organisation de l'écran

Niveau riche

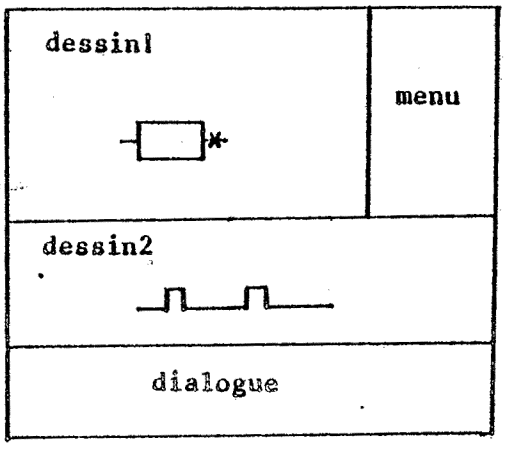
En mode de fonctionnement courant, l'utilisateur verra sur son écran trois zones:

- la zone "dessin" qui lui sert à dessiner l'application en cours;
- la zone "menu" qui lui sert à sélectionner les actions;
- la zone "dialogue" qui lui permet de recevoir les différents messages.

On peut associer, à chaque zone, un fond de couleur différente assez pâle, pour ne pas attirer l'attention de l'utilisateur.



L'utilisateur peut également diviser sa zone de dessin en plusieurs zones. Ceci peut, par exemple, arriver à la simulation lorsqu'il veut visualiser à un certain endroit de l'écran, ce qui se passe sur un fil:



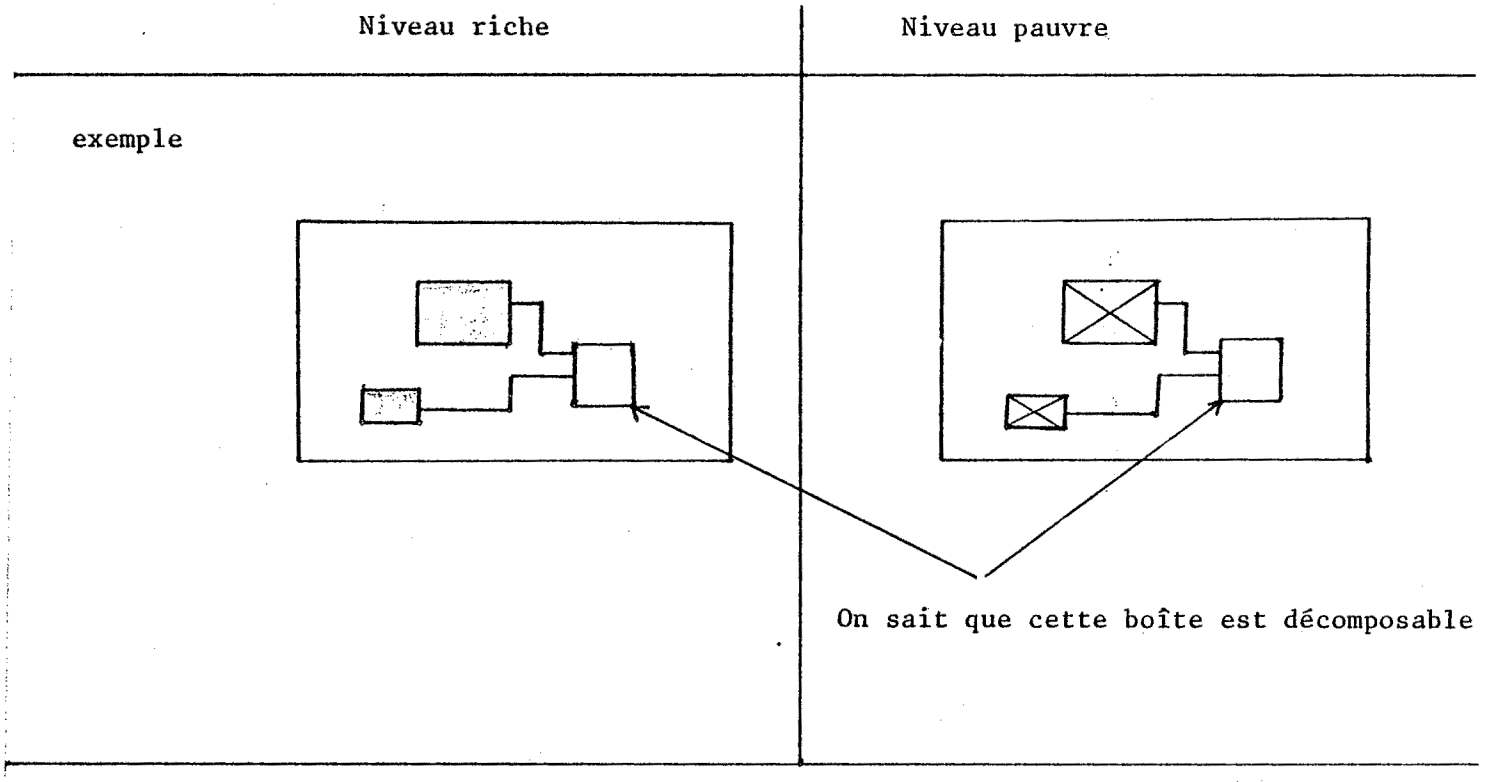
On peut de la même manière associer à chaque "sous-zone" une couleur dérivée de la couleur de la zone.

Niveau pauvre

Ici, la couleur ne représente rien de fondamental. Elle est simplement un agrément pour l'utilisateur. Un simple trait marquant les frontières entre les différentes zones peut se substituer dans ce cas-là à l'utilisation de la couleur.

2) Entrée d'une nouvelle description

	Niveau riche	Niveau pauvre
Contour de la boîte	La couleur associée au contour de la boîte représente le niveau de description selon les conventions adoptées dans a).	On peut visualiser de manière textuelle le niveau de description dans un coin de la description.
Pattes d'E/S	Associer une couleur à chaque type. Un sous-type peut être représenté de la même couleur moins saturée.	Rien.
Connexions	Les connexions sont tracées avec la couleur du type de l'information qui circule sur cette connexion.	Rien. L'utilisateur a tout de même un message s'il se trompe en essayant de relier des pattes de types différents.
Placement des boîtes prédéfinies	Chaque boîte apparaît avec : . la couleur de son contour correspondant au niveau de description; . la couleur de ses pattes correspondant aux types des informations circulant sur chacune d'elles; . l'intérieur entièrement colorié avec la couleur du contour si la boîte est feuille de l'arbre des imbrications.	Si une boîte est une feuille de l'arbre des imbrications, elle apparaît avec une croix à l'intérieur.



Disposer de la couleur est donc un avantage considérable. Nous voyons que cela ne sert pas seulement à agrémenter des schémas. La sémantique associée à la couleur est dans certains cas très importante et évite de surcharger les dessins avec du texte.

CHAPITRE IV

PRINCIPES ET SPECIFICATIONS D'UN GENERATEUR D'EDITEURS

CHAPITRE IV

PRINCIPES ET SPECIFICATIONS D'UN GENERATEUR D'EDITEURS

Cette thèse avait pour objectif de présenter un nouvel éditeur, spécifique à l'application CASCADE. Les possibilités de décrire le comportement d'un système graphiquement, d'entrer des spécifications à différents niveaux de langage et d'obtenir une traduction dans un langage textuel en font un éditeur original.

Cependant, nous sommes conscients que cet éditeur ne pourra que difficilement servir à une application autre que CASCADE. Nous ne comptons plus le nombre d'éditeurs spécialisés développés ces dernières années. Nous ne prendrons comme exemple que le système UNIX qui offre à ses utilisateurs un grand nombre d'éditeurs consacrés chacun à une application précise. Citons, par exemple, vi, ex, emacs, emin, pour éditer du texte, troff pour formatter du texte, eqn pour entrer des formules mathématiques, tbl pour les tableaux, ...

Ce phénomène de "dispersion" est peut-être dû au manque de travaux visant une formalisation des éditeurs. Evidemment, ce problème est difficile puisque les applications se développent dans de nombreuses voies différentes (MeV82, VDa71).

Ces dernières années pourtant, on constate une tendance à une plus grande intégration des différents éditeurs. Des produits tels que BRAVO (Lam78) ou SMALLTALK (Gor83) développés à Xerox PARC en sont la preuve. Il y a une volonté très nette d'intégration des différents éditeurs spécialisés en un seul éditeur. Des études faites à l'INRIA ont également le même objectif. (Hul83)

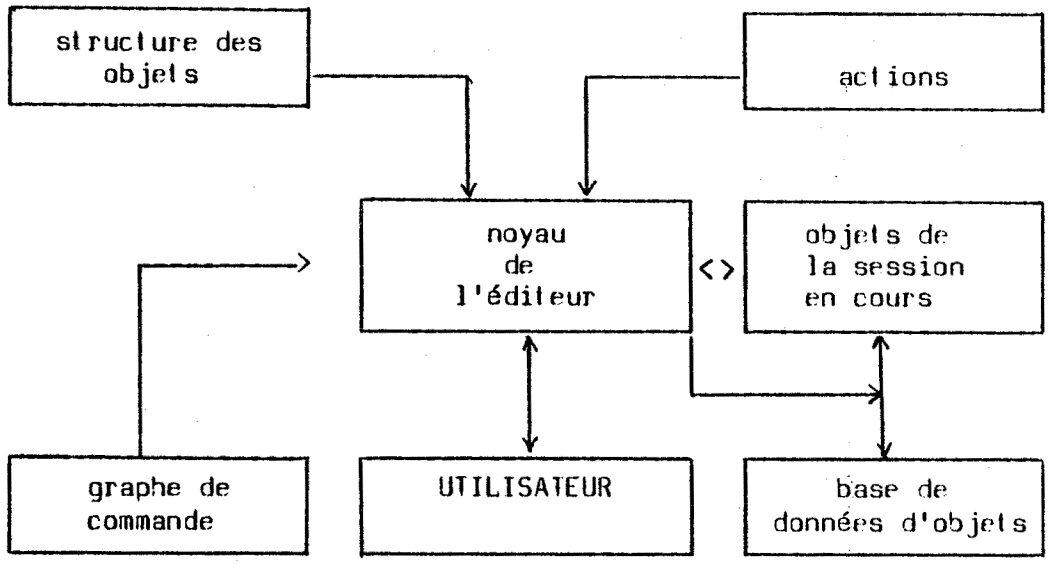
Nous voyons donc que les orientations dans le domaine des éditeurs ont changé et que le produit vers lequel on tend est plutôt un outil dans lequel l'utilisateur pourrait entrer tous les types d'informations que l'on peut trouver sur des feuilles de papier. On pourrait par exemple, entrer du texte avec des formules mathématiques, des tableaux et surtout des images (notamment depuis l'apparition des écrans "bit-map" et de bases de données permettant de stocker un grand nombre d'informations) (Tig83)

Ce chapitre est une réflexion sur une autre approche d'unification d'éditeurs qui permettrait à l'utilisateur de spécifier lui-même l'éditeur dont il a besoin (Mar84). Pour

cela, nous dégagerons tout d'abord les points communs entre les différents éditeurs pour définir un schéma général d'organisation. Puis, nous spécifierons un outil général permettant la définition d'éditeurs spécialisés.

IV.1 STRUCTURE GENERALE DES EDITEURS CONSIDERES

Parmi les points communs des différents éditeurs étudiés, la structure nous paraît en être un fondamental. En cherchant une structure commune, nous avons remarqué que tous les éditeurs que nous considérons peuvent se ramener à l'organisation suivante:



Lorsque l'on débute une session, le graphe de commande est dans un certain état initial qui représente un contexte. Les actions possibles sont définies par les commandes du contexte. L'utilisateur choisit une des commandes, ce qui aura pour effet l'exécution d'actions associées stockées dans la base de données d'actions. Les actions travaillent sur l'objet en cours de traitement, dont la structure est décrite dans "structure des objets". Dans le cas d'une lecture ou d'une écriture, il y a accès à la base de données d'objets. Une fois l'action effectuée, l'utilisateur se trouvera dans un autre contexte (éventuellement le même) qui sera positionné selon l'action qui s'est déroulée. On itère alors le processus jusqu'à rencontrer la commande de fin d'exécution.

* Description d'un éditeur

Pour chaque éditeur, nous aurons à spécifier:

- la structure des éléments que l'on manipulera dans l'éditeur;
- l'organisation des commandes de l'éditeur;
- les actions de base autorisées par l'éditeur.

IV.2 DESCRIPTION DE LA STRUCTURE

Considérons, tout d'abord, les éléments qui seront manipulés dans les éditeurs et voyons comment les décrire.

IV.2.1 TYPE DE STRUCTURE CHOISI

Dans chaque éditeur, les objets ont leur propre structure. La plupart du temps, dans les éditeurs "texte", les objets sont structurés en texte, paragraphes, phrases, mots et lettres. La notion de lignes existe seulement au niveau de l'impression, et n'intervient pas dans la structure d'un texte. Dans les éditeurs graphiques, les objets sont également structurés: on compose un objet à l'aide d'éléments de base. L'objet ainsi construit peut alors faire partie d'un autre objet plus global, etc.

Dans les deux cas, les objets manipulés sont structurés sous forme arborescente. D'une manière générale, nous pouvons dire qu'en conception, nous travaillons soit par décomposition, soit par composition.

Par décomposition, la démarche du concepteur consiste à partir d'un objet qu'il peut décomposer en plusieurs sous-objets, eux-mêmes décomposables... jusqu'à arriver aux objets primitifs (conception descendante).

Par composition, l'utilisateur part d'éléments de base, construit grâce à eux des éléments plus complexes, qui pourront à leur tour être utilisés pour construire des objets de complexité encore plus grande... jusqu'à obtenir l'objet désiré (conception ascendante).

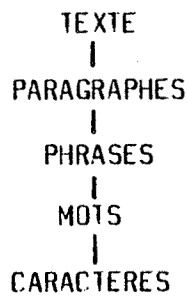
C'est pourquoi nous avons choisi de représenter la structure des objets manipulés par les éditeurs sous forme arborescente.

IV.2.2 NOTION DE REPETITION

Pour chaque éditeur, il faut non seulement savoir que les éléments qu'il manipule sont ainsi structurés mais il faut également imposer quelques contraintes supplémentaires sur la hiérarchie des objets les uns par rapport aux autres: par exemple, un texte est composé de plusieurs paragraphes, eux-mêmes composés de plusieurs phrases...

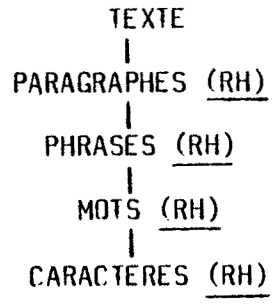
Il faut pouvoir dire qu'une phrase ne peut pas être considérée comme un texte, c'est-à-dire qu'elle ne peut pas, par exemple, contenir de paragraphes.

Cela nous amène naturellement à la notion de modèle de structures de données manipulées par l'éditeur. Dans l'exemple décrit ci-dessus (exemple 1), le modèle serait:

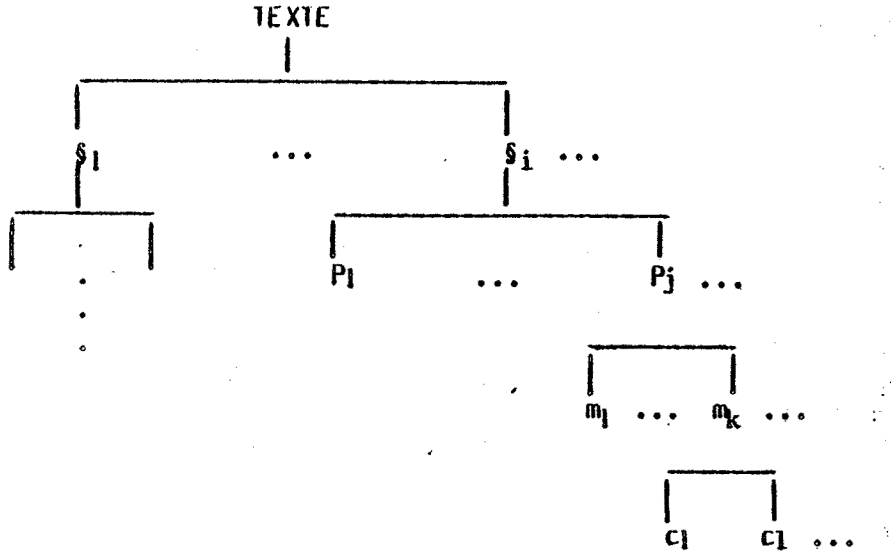


On remarque qu'il manque une notion dans ce modèle: la répétition. En effet, on veut, dans notre exemple, pouvoir dire que l'on a plusieurs paragraphes possibles dans un texte ou plusieurs phrases possibles dans un paragraphe. On veut également pouvoir dire que l'on n'a qu'un seul texte.

Nous noterons cela de la manière suivante:



ce qui est équivalent à

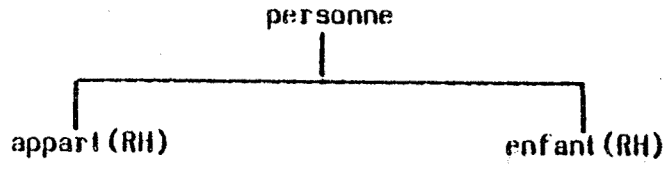


En traitant cet exemple, ces notions semblent suffire pour décrire ce que l'on désirait.

Considérons, à présent, l'exemple suivant (exemple 2):

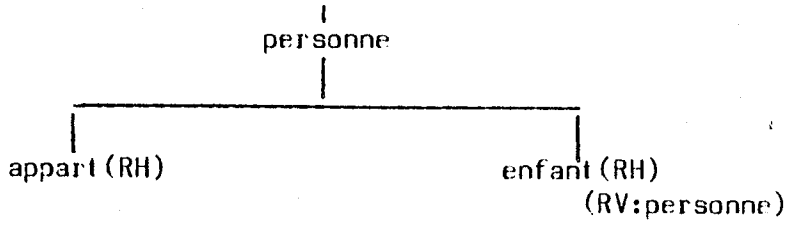
Nous avons une liste de personnes ayant chacune un certain nombre d'appartements et un certain nombre d'enfants.

D'après ce qui a été décrit ci-dessus, le modèle associé serait le suivant:

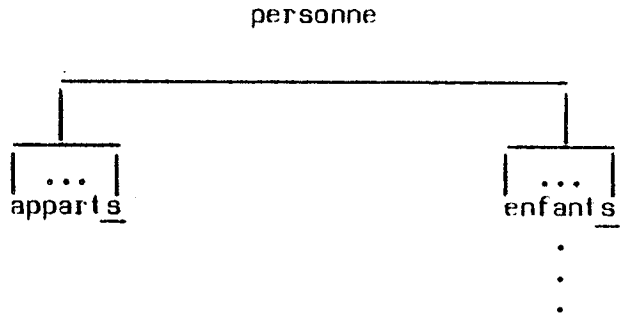


Un autre problème se pose: les enfants sont eux-mêmes des personnes qui peuvent à leur tour avoir des appartements et des enfants.
C'est encore une notion de répétition dont on a besoin, mais, contrairement à la précédente qui était une répétition horizontale (RH), celle-ci est une répétition verticale (RV).

Le modèle sera alors le suivant:



ce qui est équivalent à

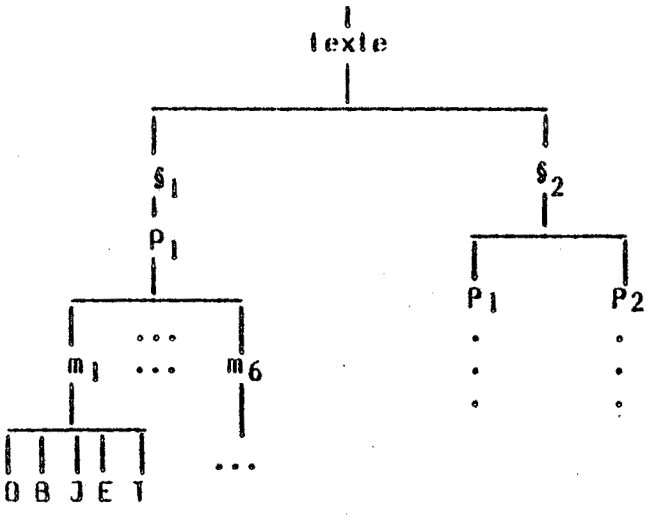


Un modèle peut être considéré comme un ensemble de règles qui permettent de construire un arbre particulier que l'on appellera occurrence de l'objet.

Ainsi, dans l'exemple 1, l'objet:

OBJET: Demande de documentation; TITRE: Présentation générale du projet TIGRE Janvier 1983

sera représenté par l'occurrence:



IV.2.3 ATTRIBUTS

Nous allons à présent introduire la notion d'attributs attachés aux noeuds des arbres: un attribut représente une caractéristique attachée à un certain noeud de l'arbre.

Dans l'exemple 1, on peut avoir comme attribut attaché à chaque noeud: type de caractères (majuscules ou minuscules).

Dans l'exemple 2, un numéro de sécurité sociale peut être attaché à chaque personne (attribut de la personne).

Il arrive souvent que les fils d'un noeud aient les mêmes attributs que ce noeud, en en ayant en plus certains propres qui ne sont pas dans le noeud père.

Cette notion est déjà exposée dans le langage SIMULA avec la notion de classes préfixées, et reprise dans le langage SMALLTALK avec la notion de sous-classes: une certaine classe C1 est décrite en précisant un certain nombre d'attributs; tous les éléments appartenant à C1 auront ces attributs. Une sous-classe de C1 est caractérisée par les mêmes attributs que C1 avec en plus d'autres qui lui sont propres.

IV.2.4 LANGAGE DE SPECIFICATION DE STRUCTURES

Il faut donc définir un langage de spécification de structures. Pour cela, nous nous sommes inspirés des mécanismes de description des types en PASCAL (Wir74).

Chaque noeud du modèle appartient à une classe décrite par son nom, ses attributs et ses fils.

```
Classe N
   |
   |  Champs relatifs aux attributs.
   |
   |
   |  Fils.
   |
Fin_classe;
```

Les attributs ont un type quelconque alors que les fils sont forcément de type classe.

La notation A* précise qu'il peut y avoir un nombre indéterminé d'objets de type A (éventuellement 0) et la notation A^+ précise qu'il y a au moins un objet du type A.

La notation x:y donne le type y à l'objet de nom x dans le cas d'un attribut et précise que l'objet x appartient à la classe y dans le cas d'un fils.

Exemple 1 se traduira par:

```
Classe texte
   |    type_écriture:  (majuscule, minuscule);
   |    liste_paragraphe:  paragraphe+;
Fin_classe;

Classe paragraphe
   |    type_écriture:  (majuscule, minuscule);
   |    liste_phrase:  phrase+;
Fin_classe;
 etc...
```

où (majuscule, minuscule) décrit le type énuméré qui contient les deux valeurs: 'majuscule' et 'minuscule'.

Exemple 2 se traduira par:

```
Classe PERSONNE
   |    attribut éventuel (n securite sociale...  );
   |    liste_appart:  APPART*;
   |    liste_enfant:  ENFANT*;
Fin_classe;

Classe APPART
   |    attribut éventuel (adresse, superficie...  );
Fin_classe;

Classe ENFANT dérive de PERSONNE;
   |    attributs propres de l'enfant;
Fin_classe;
```

REMARQUES:

La répétition horizontale (RH) se traduira grâce à la notion d' ou de + qui indiquent la répétition d'une structure de même type.

"Classe ENFANT dérive de PERSONNE" annonce la description de la classe enfant qui est une sous-classe de personne. Ceci sous-entend que tous les éléments de la classe enfant auront tous les attributs et les fils de la classe personne plus ceux de la classe enfant.

Par convention, les attributs de la classe (modèle) sont définis avant ceux propres à la sous-classe.

IV.3 L'ORGANISATION DES COMMANDES

Après avoir décrit la structure des éléments manipulés dans l'éditeur, voyons maintenant comment décrire le séquencement des commandes de l'éditeur.

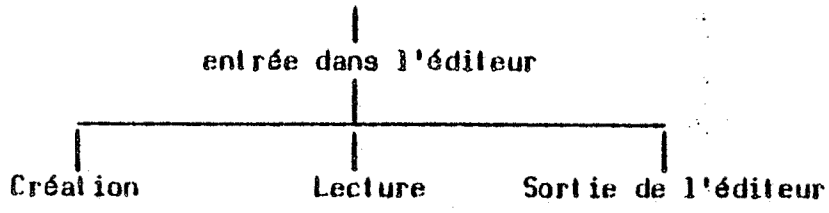
IV.3.1 TYPE DE STRUCTURE

Pour définir le fonctionnement de son éditeur, chaque concepteur d'éditeurs se doit de définir l'enchaînement possible des actions.

Ce séquencement peut se décrire par un arbre.

Un noeud de cet arbre représente un contexte dans lequel les seules commandes exécutables sont celles décrites par les noeuds fils de ce contexte.

Exemple:



Si, quand on commence l'exécution d'une édition, le noeud courant de l'arbre de contrôle est le noeud "entrée dans l'éditeur", trois possibilités se présentent à l'utilisateur:

- Création d'objets;
- Lecture d'objets déjà existants en vue de modifications;
- Sortie de l'éditeur.

Il faut également associer à chaque noeud de l'arbre ainsi décrit les types de paramètres que l'utilisateur peut (paramètres facultatifs) ou doit (paramètres obligatoires) donner pour que l'action associée à la commande puisse s'effectuer.

D'autre part, il est nécessaire de décrire des notions autres que celle de contexte. Par exemple, il faut pouvoir préciser que l'on peut répéter certaines commandes un nombre indéterminé de fois sans changer de contexte jusqu'à rencontrer la commande "exit" ou bien au contraire que l'on ne peut exécuter qu'une fois une des actions proposées, etc.

Cela est réalisé en associant à chaque noeud de l'arbre de contrôle l'une des quatre primitives: environnement, commande, choix ou feuille.

"environnement" permet de préciser l'ensemble des commandes activables à cet instant. On peut répéter autant de commandes que l'on veut en restant dans le même contexte jusqu'à activer une commande de sortie.

"commande" permet de préciser que l'on exécute une action complète associée à une commande. On ne l'exécute qu'une fois et l'on se retrouve dans l'environnement d'appel.

"choix" permet d'exécuter une action parmi un ensemble donné d'actions. Cette primitive a été introduite pour que les commandes ne soient pas trop rigides. (On interdit d'avoir des environnements dans des commandes).

"feuille" permet de préciser que l'action associée à cette commande est terminale.

IV.3.2 DESCRIPTION DE CETTE STRUCTURE

Le langage de description qui permet cela existe déjà. Il a été développé dans le cadre du projet CASCADE. La grammaire complète de ce langage permettant de décrire toute l'organisation des commandes et même les récupérations en cas d'erreur est décrite dans (DLU84).

IV.4 LES ACTIONS

IV.4.1 FORME GENERALE D'UNE ACTION

Dans chaque éditeur, nous avons un ensemble d'actions entièrement déterminé. Le problème est de définir ce que doit faire une action.

Selon notre approche, chaque action est associée à une commande (cf. organisation en IV.3) et n'agit que sur la structure de données de l'application.

IV.4.2 ACTION SUR LA STRUCTURE

Une action agit

- soit sur les attributs de l'arbre;
- soit sur la structure de l'arbre.

A chaque action, on associera une procédure à suivre qui indiquera toutes les modifications à faire sur la structure. Ces procédures pourront faire partie d'une base de données de procédures précompilées une fois l'éditeur décrit.

Remarquons que certains paramètres de ces procédures ont déjà été fournis lors de la désignation de la commande associée à l'action.

Exemple:

Considérons les clients d'une banque décrits de la manière suivante:

```
classe client
    | nom_client:chaîne;
    | num_compte:integer;
    | liste_débits_automatiques:deb_auto*;
Fin_classe;

classe deb_auto
    | valeur:integer;
    | date_débit:date;
Fin_classe;
```

où "chaîne" et "date" sont des types prédéfinis.

Soit l'action suivante à effectuer:
"Enregistrement d'un nouveau client"

Procédure à suivre pour cela:

```
Demander ("nom du client",NCL);
Demander ("No de compte du client",NCOMPTE);
Demander ("Avez-vous des débits automatiques",A_DEBIT_AUTO);
ENUM_DEBIT:=nil;
Créer (CLIENT);
  CLIENT↑.nom_client:=NCL;
  CLIENT↑.num_compte:=NCOMPTE;
  Tantque A_DEBIT_AUTO faire
    Demander ("date de débit automatique",D_DEB_AUTO);
    Créer (DEB_AUTO);
      DEB_AUTO↑.date:=D_DEB_AUTO;
      DEB_AUTO↑.autre_débit:=ENUM_DEBIT;
      ENUM_DEBIT:=DEB_AUTO;
    Demander ("Avez-vous d'autres débits automatiques",
                                          A_DEBIT_AUTO);
  Fin_tantque;
  CLIENT↑.liste_débits_automatiques:=ENUM_DEBIT;
```

REMARQUES:

* Les actions sont traduites par une suite d'instructions très proches du langage PASCAL.

* Demander ("...", variables) est une primitive qui permet d'imprimer un message et d'avoir la réponse dans la (ou les) variable(s) qui suivent.

* Créer permet de créer un objet dont le type est précisé dans la partie structure.

* La notation CLIENT↑. nom_client indique qu'il s'agit du champ nom_client qui fait partie de l'objet pointé par CLIENT.

* Les paramètres NCL, NCOMPTE auraient pu être fournis lors de l'appel de la commande d'enregistrement d'un nouveau client.

IV.5 LA PORTEE DES ACTIONS

IV.5.1 DEFINITION

Dans tous les langages de programmation, il existe la notion de "portée de déclarations de variables". Une variable déclarée dans une procédure ne peut être utilisée que dans la procédure, par contre, une variable déclarée en tête du programme (variable globale) peut être utilisée dans n'importe quelle partie du programme. Pour plus de précisions sur les notions de "portée des variables" et de "blocs de programmation", le lecteur pourra se référer à un quelconque ouvrage sur la méthodologie de programmation. Citons, par exemple (Luc78).

Nous nous intéressons ici à la notion de "portée des actions". Dans la plupart des éditeurs, les commandes agissent sur un objet simple (un caractère par exemple dans le cas d'un éditeur texte) ou bien sur un objet composé (une phrase = ensemble de caractères dans le même exemple).

Exemple:

Dans l'éditeur de texte TED (intégré dans le système MULTICS), la commande s/titi/toto agit sur la ligne courante du texte et remplace toutes les occurrences de titi par toto, tandis que 1,$s/titi/toto agit sur la totalité du texte et remplace toutes les occurrences de titi par toto. Ici, la portée de l'action est définie en tête de la commande et on prend par défaut la ligne courante.

Nous n'avons pas encore abordé ce problème dans les paragraphes précédents. En fait, il faudra s'assurer que l'on connaît bien cette portée avant toute activation d'une action.

Deux solutions sont envisageables:

1. On précise la portée de l'action avant chaque action.
2. On a une notion de portée courante.

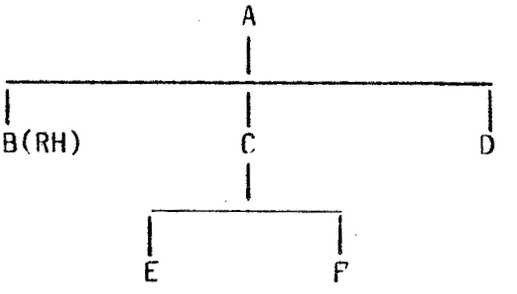
La deuxième solution semble préférable pour l'utilisateur puisqu'elle permet de ne pas redéfinir à chaque fois la même chose.

Nous pourrons pour mettre en oeuvre cette solution avoir des primitives spéciales qui permettent de redéfinir la portée des opérations à chaque fois que cela est nécessaire.

IV.5.2 DESCRIPTION PROPOSEE

Le langage qui permet de décrire la portée d'une action sera à peu près le même que celui de CLOVIS (Gra83), (Mar82).

Ainsi, si nous avons l'arbre



Portée (a (b (2,3) , c (e))) indiquera que les actions suivantes se passeront sur les noeuds a , b (2ème et 3ème noeud), c et e.

La portée sera valable jusqu'à la définition d'une nouvelle portée.

Remarque: b (2,3) est une notation possible puique l'on ne sait pas à priori combien de noeuds b il y aura (voir notion de répétition horizontale plus haut).

IV.6 EXEMPLE: L'EDITEUR GRAPHIQUE DE CASCADE

IV.6.1 LE PROBLEME

Nous considérons ici un sous-ensemble de l'éditeur graphique de CASCADE et nous le décrivons en donnant la structure de données d'une part et l'enchaînement des commandes d'autre part selon les schémas énoncés dans les chapitres précédents. Un exemple de spécification d'action est également donné conformément au schéma de description d'actions.

IV.6.2 DESCRIPTION D'UNE PARTIE DE LA STRUCTURE DE L'EDITEUR

Les objets manipulés dans cet éditeur sont des descriptions. Chaque description représente un élément prédéfini ou que le concepteur a défini.

Dans l'éditeur graphique, on associe à chaque description

* un nom;

* un niveau de langage qui précise le niveau de description auquel on se trouve;

* un contour qui permet de reconnaître l'objet graphiquement;

* des éléments d'interface qui sont les emplacements physiques d'échange avec l'extérieur;

* des boîtes imbriquées qui sont les composants de l'élément que l'on décrit;

* des attributs qui décrivent quels sont les paramètres variables dans le cas de descriptions génériques;

* des connexions entre ces différents composants;

* un comportement.

Un tel objet sera spécifié par

```
classe description
    | Nom: chaîne;
    | NL: chaîne;
    | Contour: elem_contour+;
    | Eleminf: elem_interf*;
    | Bteimbr: boîtes_imbriquées*;
    | Conn: connexions*;
    | Att: attributs*;
    | Comp: comportement;
Fin_classe;
```

Chacun des noeuds

. elem_contour,

. elem_interface,

. boîtes_imbriquées,

. connexions,

. attributs,

. comportement,

doit à son tour être précisé.

Décrivons par exemple elem_contour.

Un contour est une suite de parties de contour. Chaque partie de contour est une liste de points reliés entre eux soit par des segments, soit par interpolation. Un tel objet se décrira ainsi:

```
classe elem_contour;
  |Mode:  (trait, interpolation);
  |Points_à_relier:  Suite_coord+;
Fin_classe;
```

où Suite_coord est ainsi représenté:

```
classe Suite_coord;
  |X:  integer;
  |Y:  integer;
Fin_classe;
```

Décrivons encore elem_interface.

Pour chaque élément d'interface, nous devons préciser

- son nom,
- son type,
- son sens,
- sa dimension,
- ses coordonnées relatives par rapport à la boîte à laquelle il appartient.

La description est alors:

```
classe elem_interface;
  |Nom:  chaîne;
  |Type:  chaîne;
  |Sens:  (entrée,sortie,non_directionnel,bi_directionnel);
  |Dimension:  dim*;
  |Xrel:  integer;
  |Yrel:  integer;
Fin_classe;
```

avec

```
classe dim;
  |Dimgauche:  integer;
  |Dimdroite:  integer;
Fin_classe;
```

IV.6.3 ENCHAINEMENT DES COMMANDES

L'enchaînement des commandes autorisé a été décrit par une succession de menus que l'utilisateur voit apparaître suivant les commandes qu'il choisit d'exécuter. A partir de cette spécification dont la forme convient plus spécialement à l'utilisateur (Mar83), une description complète de l'enchaînement des commandes a été décrite dans (DLU84) suivant le schéma défini dans IV.3.

IV.6.4 EXEMPLE DE SPECIFICATION D'UNE ACTION DE L'EDITEUR

Soit par exemple l'action "déplacer un élément d'interface".

Cette action est décrite dans un langage comprenant un certain arbre de primitives (ici Demander, Afficher):

```
Demander ("Montrer l'élément à déplacer", ELEMENT);
Demander ("Montrer le nouvel emplacement", EMPL_X, EMPL_Y);
                        .
                        .
                        .
            Suite d'instructions pour calculer
            EMPL_X et EMPL_Y par rapport au
            barycentre de la boite à laquelle
            appartient l'élément à déplacer.
            Le résultat se retrouve dans
            EMPL_REL_X et EMPL_REL_Y.
                        .
                        .
                        .
ELEMENT .  Xrel:= EMPL_REL_X;
ELEMENT .  Yrel:= EMPL_REL_Y;
Afficher (ELEMENT);
```

Remarques:

* Afficher est une "macro-instruction" qui permet de visualiser ses paramètres sur l'écran. Elle est dépendante du logiciel graphique de base utilisé. Dans le cas de l'éditeur graphique de CASCADE, c'est le logiciel CLOVIS qui réalise cette action.

* ELEMENT, EMPL_X, EMPL_Y peuvent être fournis en paramètres de la commande.

IV.7 UTILISATION D'UN EDITEUR DEFINI COMME CI-DESSUS

Une fois l'éditeur complètement décrit, c'est-à-dire une fois que l'on a donné la structure des éléments qui seront manipulés (cf IV.2), le contrôle (cf IV.3), les actions autorisées (cf IV.4), et les portées d'actions (cf IV.5), le fonctionnement est le suivant:

- Nous positionnons le contexte courant grâce à un certain contexte initial.
- Une proposition de choix possibles est alors faite à l'utilisateur. Ce sont les commandes activables dans le contexte courant.

- L'utilisateur précise son choix.

- On exécute l'action associée dans la base de données d'actions avec la portée décrite.

- On se retrouve dans un nouveau contexte pour la suite des opérations.

On itère ces cinq étapes jusqu'à activer la commande "sortie de l'éditeur".

CONCLUSION

Bilan de l'expérience

Le système CASCADE est un système ambitieux qui met en oeuvre un grand nombre de techniques informatiques. Certaines sont très connues (compilation...), d'autres sont encore très récentes (logique temporelle...).

Travailler sur un tel projet a été intéressant à plus d'un titre: les communications entre les partenaires européens, l'intégration des différentes parties du système écrites par chacun d'eux, la nécessité de portabilité pour pouvoir tester les programmes sur différents "sites" sont autant de problèmes difficiles à résoudre mais très importants dans un projet de cette taille.

L'expérience retirée nous permet d'entrevoir une organisation différente (et plus stricte) par rapport aux "petits" projets auxquels nous avions l'habitude d'être confrontés.

Proposition pour un véritable prototype de l'éditeur graphique

Actuellement, l'éditeur graphique de CASCADE ne permet d'entrer des descriptions qu'au niveau "portes logiques". En effet, seule la partie structurelle des descriptions peut être entrée graphiquement et traduite en CASCADE.

Les spécifications données au chapitre III devraient permettre assez rapidement d'entrer le comportement d'une description graphiquement. Il deviendra alors possible de décrire des descriptions graphiquement à n'importe quel niveau de langage.

Extensions envisageables à plus long terme

Un travail de réflexion supplémentaire doit être entrepris avant toute mise en oeuvre d'une programmation du générateur d'éditeurs décrit dans le chapitre IV. De nombreux points restent à éclaircir, surtout en ce qui concerne la description des actions. Etant donnée l'importance attachée actuellement à ce genre d'outils et à l'intérêt qu'il susciterait s'il existait, nous espérons continuer ce travail en collaboration avec les différents organismes concernés.

BIBLIOGRAPHIE: Introduction.

* Bor81 * D. Borrione

Langages de description de systèmes logiques.
Propositions pour une méthode formelle de définition.
Thèse d'état, Grenoble.

Juil 1981

* Feu83 * M. Feuer

VLSI Design Automation: An Introduction
Proc. IEEE, Vol71, N° 1

Jan. 1983

* Lip77 * G.J. Lipovski

Hardware Description Languages: Voices from the Tower of Babel
Computer, Vol 10, n° 6

June 1977

* Mer83 * J. Mermet

Conception Assistée de Systèmes & Circuits Analogiques et Digitaux Electoniques: CASCADE.
Bulletin spécial de Micado.

Janv. 1983

* PBB80 * **R. Piloty, M. Barbacci, D. Borrione, D. Dietmeyer, F. Hill, P. Skelly**
CONLAN- A formal construction method for hardware description languages
3 papers in NCC 1980, AFIPS Conference Proceedings, Vol 49, pp 209-236
1980

* PBB83 * **R. Piloty, M. Barbacci, D. Borrione, D. Dietmeyer, F. Hill, P. Skelly**
CONLAN Report
Lecture Notes in Computer Science n°151,
Springer Verlag
1983

* PiB82 * **R. Piloty, D. Borrione**
The CONLAN Project: Status and Future Plans
Proc 19th DAC pp 202-212
June 1982

BIBLIOGRAPHIE: Chapitre I.

* EEK79 * J. Encarnacao, G. Enderle, K. Kansy, F.J. Prester
GKS 79: Proposal of a standard for a Graphic Kernel System.
Congrès EUROGRAPHICS, Bologna.
Sept 1979

* EPS77 * R. Eckert, F.F. Prester, E.G. Schlechtendahl, P. Wisskirchen
Functional Description of the Graphical Core System GKS as a Step Towards Standardisation
Report from FNI Working Group 5.9, first draft, juillet 1977

* Gra83 * J.F. Grabowiecki
Doc utilisateur de CLOVIS. Rapport interne.
Juin 1983

* GSP79 * Status report of the Graphics Standard Planning Commitee.
Computer graphics, SIGGRAPH-ACM, Vol 13, n° 3
Aug 1979

* Gue76 * **R.A. Guedj**

Report on the IFIP W.G. 5.2. Workshop on "Methodology in Computer Graphics"

juil 1976

* HP82 * **Hewlett Packard**

D.G.L.1000: Manuel d'utilisation

1982

* Hvi79 * **H. Hvistendahl**

I.G.L.: A structured language for interactive graphics.

Congrès EUROGRAPHICS, Bologna.

Sept 1979

* ISO82 * **Information processing Graphical Kernel System (G.K.S.).**

Functional description.

Draft International Standard ISO/DIS 7942

Nov 1982

* LLM76 * A. Leduc-Leballeur, M. Lucas, F. Martinez

GRIGRI: Logiciel de base pour l'utilisation des consoles de visualisation du CICG.

Note technique n° 49.

Dec 1976

* LLM78 * A. Leduc-Leballeur, M. Lucas, F. Martinez

Conception et réalisation d'un logiciel graphique interactif indépendant du contexte d'utilisation. Le logiciel de base GRIGRI. .

Revue RAIRO Informatique, Vol 12, n°2.

1978.

* Luc77 * M. Lucas

Contribution à l'étude des techniques de communication graphique avec un ordinateur. Elements de base des logiciels graphiques interactifs.

Thèse d'état, Grenoble.

Dec 1977.

* Mar82 * F. Martinez

Vers une approche systématique de la synthèse d'image. Aspects logiciel et matériel.

Thèse d'état, Grenoble.

Nov 1982

* Tek82 * **Société TEKTRONIX**

PLOT10 Interactive Graphics Library.
User's manual.

Oct 1982

* The72 * **F. Theron**

Sur le programme EUCLID: Création, manipulation et visualisation de formes tridimensionnelles dans un langage géométrique à génération dynamique.
Thèse de 3ème cycle, Paris.

1972

BIBLIOGRAPHIE: Chapitre II.

* BMV83 * D. Borrione, J. Mermet, M. Vergniault
Gate level language in CASCADE: POLO.
Rapport de projet C.E.E. n° bHDL-IMAG-SS-001
Dec 1983

* Bor81 * D. Borrione
Langages de description de systèmes logiques.
Propositions pour une méthode formelle de définition.
Thèse d'état, Grenoble.
Juil 1981

* Bou84 * A. Boulze
Extension du logiciel CLOVIS pour utilisation sur HELIOS
Rapport CNAM
Oct. 1984

* Bre83 * Y. Bressy
Définition et implémentation des différentes notions et des mécanismes du langage CASCADE.
Rapport de projet C.E.E. n° bHDL-IMAG-SR-001
Nov 1983

* Equ82 * **Equipe C.A.O. de l'ENSIMAG**

Langage CASCADE.
(Conception Assistée de Systèmes et de Circuits Analogiques et Digitaux Electroniques)

Mai 1982

* Gui84 * **M.T. Guichard**

Choix et réalisation d'un algorithme de routage pour l'éditeur graphique de CASCADE.
Rapport de D.E.A. Grenoble

Sept 1984

* Lee61 * **L.Y. Lee**

An algorithm for path connections and its application.
I.E.E.E. Trans. Electro. Comp., Vol EC-10

Sept 1961

* Mar83a * **J.C. Marty**

CASCADE graphical editor. Specification of the user's interface.
Rapport de projet C.E.E. n° GRED-IMAG-WD-1

May 1983

* Mar83b * J.C. Marty

Spécification du scénario d'utilisation de l'éditeur graphique et de la structure de données utilisée.
Rapport de projet C.E.E. n° aGRS-IMAG-SS-001
Août 1983

* Mar84a * J.C. Marty

Documentation système du programme de l'éditeur graphique.
Rapport interne.
Août 1984

* Mar84b * J.C. Marty

Manuel d'utilisation de l'éditeur graphique de CASCADE.
Rapport interne.
Août 1984

* Mei71 * J.P. Meinadier

Structure et fonctionnement des ordinateurs.
Larousse, Paris
1971

* Mer73 * **J. Mermet**

Etude méthodologique de la Conception Assistée par Ordinateur des systèmes logiques: CASSANDRE.
Thèse d'état, U.S.M.G., Grenoble

Avril 1973

* Mer83 * **J. Mermet**

Conception Assistée de Systèmes & Circuits Analogiques et Digitaux Electoniques: CASCADE.
Bulletin spécial de Micado.

Janv. 1983

* Que81 * **J.P. Queille**

The CESAR system: an aided design and certification system for distributed applications.
Proceedings of the 2nd International Conference on distributed computing systems.

April 1981

* Ser82 * **G. Serrero**

Implantation symbolique automatisée de circuits intégrés.
Thèse de docteur-ingénieur, I.N.P.G.

Mars 1982

Bibli n°11

* Sif79 * J. Sifakis

Le contrôle des systèmes asynchrones: Concepts, propriétés, analyse statique.

Thèse d'état, université de Grenoble.

Juin 1979

* Wir74 * N. Wirth

PASCAL, User manual and report.

2nd edition, Springer-Verlag, New-York

1974

BIBLIOGRAPHIE: Chapitre III.

* BMB83 * **D. Borrione, J. Mermet, (IMAG)**
G. Battistoni, P. Prinetto (Politecnico di Torino)
A time profile description language for system specification and simulation.
Rapport de recherche n° 406, IMAG
Nov. 1983

* Bor81 * **D. Borrione**
Langages de description de systèmes logiques.
Propositions pour une méthode formelle de définition.
Thèse d'état, Grenoble.
Juil 1981

* BoG80 * **D. Borrione, J.F. Grabowiecki**
Manuel d'utilisation LASSO
Août 1980

* Bre83 * **Y. Bressy**
Définition et implémentation des différentes notions et des mécanismes du langage CASCADE.
Rapport de projet C.E.E. n° bHDL-IMAG-SR-001
Nov 1983

* Bre84a * Y. Bressy

Réalisation des niveaux CASSANDRE et LASCAR du langage CASCADE.

Rapport de projet C.E.E. n° bHDL-IMAG-SR-003

Janv 1984

* Bre84b * Y. Bressy

Réalisation du niveau LASSO du langage CASCADE.

Rapport du projet C.E.E. n° bHDL-IMAG-SR-002

Janv. 1984

* Dec84 * E. Decamp

Un modèle pour le raisonnement temporel.

Rapport de recherche n° 456, IMAG

Aout 1984

* Equ82 * Equipe C.A.O. de l'ENSIMAG

Langage CASCADE.

(Conception Assistée de Systèmes et de Circuits Analogiques et Digitaux Electroniques)

Mai 1982

* LeF84 * **C. Le Faou**

Ensemble de tests pour un simulateur CASCADE.
Rapport interne, IMAG

Avril 1984

* Sif79 * **J. Sifakis**

Le contrôle des systèmes asynchrones.
Concepts, priorités, analyse statique.
Thèse d'état, Université de Grenoble.

Juin 1979

* TI77 * **Texas Instruments:**

The TTL Data Book for Design Engineers.
2nd edition.

1977

BIBLIOGRAPHIE: Chapitre IV.

* BBK82 * T. Berk, L. Brownston, A. Kaufman
A new color-naming system for graphics languages.
I.E.E.E. CG&A
May 1982

* DLU84 * B. David, S. Lafont, P. Uvietta
"CASCADE User Interface Management System"
Rapport C.E.E., nSYC-IMAG-PR-002
Janvier 1984

* GoR83 * A. Goldberg, D. Robson
"SMALLTALK-83, The language and its implementation"
Ed. Addison Wesley, 3 volumes
1983

* Gra83 * J.F. Grabowiecki
"Description fonctionnelle de CLOVIS"
Rapport interne IMAG.
Janvier 1983.

* Hul83 * **J.M. Hullot**

"CEYX, a multiformalism programming environment."
Rapport INRIA n°210

Septembre 1983

* JoG78 * **G.H. Joblove, D. Greenberg**

Color spaces for computer graphics.
Computer Graphics, (SIGGRAPH 78 Proc.)
Vol 12, n°3

Aug 1978

* Lam78 * **B.W. Lampson**

"BRAVO manual"
XEROX Palo Alto Research Center

Nov. 1978

* Luc78 * **M. Lucas**

"Algorithmique et représentation des données".
Université Scientifique et Médicale de Grenoble.

Octobre 1978

* Mar82 * F. Martinez

"Vers une approche systématique de la synthèse d'images. Aspects logiciel et matériel".
Thèse d'état

Novembre 1982.

* Mar83 * J.C. Marty

"CASCADE graphics editor, Specification of the user's interface".
Rapport C.E.E., GRED-IMAG-WD

May 1983.

* Mar84 * J.C. Marty

"Principes et spécifications d'un générateur d'éditeurs."
Rapport de recherche n°428, IMAG

Fevrier 1984

* MeV82 * N. Meyrowitz , A. Van Dam

"Interactive editing systems"
"Computing Surveys"

September 1982

* Tig83 * **TIGRE.**

" Présentation générale du projet TIGRE.
Rapport de recherche n°1".
C.I.I. Honeywell Bull. Centre de recherche de Grenoble
Janvier 1983.

* VDa71 * **A. Van Dam , D.E. Rice**

"On line-text editing: a survey"
"Computing Surveys"
September 1971

* Wir74 * **N. Wirth**

"PASCAL, User manual and report"
2nd edition, Springer-Verlag, New-York
1974

AUTORISATION de SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 avril 1974,

VU le rapport de présentation de Monsieur J. MERMET, Maître de recherche

Monsieur MARTY Jean-Charles

est autorisé à présenter une thèse en soutenance pour obtenir le titre de DOCTEUR de TROISIEME CYCLE, spécialité "Informatique".

Fait à Grenoble, le 2 octobre 1984

Le Président de l'I.N.P.-G

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,