Une approche de l'édition structurée des documents

Vincent Quint

▶ **To cite this version:**

Vincent Quint. Une approche de l'édition structurée des documents. Autre [cs.OH]. Université Joseph-Fourier - Grenoble I, 1987. Français. NNT : . tel-00010612

HAL Id: tel-00010612
https://theses.hal.science/tel-00010612

Submitted on 13 Oct 2005

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

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

THESE

Présentée à

L'UNIVERSITE SCIENTIFIQUE, TECHNOLOGIQUE ET MEDICALE DE GRENOBLE

pour obtenir le grade de
docteur es-Sciences Mathématiques

par

Vincent QUINT

ooooo

UNE APPROCHE DE L'EDITION STRUCTUREE DES DOCUMENTS

ooooo

Thèse soutenue le 4 Mai 1987 devant la commission d'examen.

C. DELOBEL	**Président**
C. BOITET	
G. CORAY	
G. KAHN	**Examinateurs**
S. KRAKOWIAK	

UNIVERSITE SCIENTIFIQUE TECHNOLOGIQUE ET MEDICALE DE GRENOBLE

Président de l'Université :
M. TANCHE

Année Universitaire 1986 - 1987

MEMBRES DU CORPS ENSEIGNANT DE SCIENCES ET DE GEOGRAPHIE

PROFESSEURS DE 1ère Classe

ARNAUD Paul	Chimie Organique
ARVIEU ROBERT	Physique Nucléaire I.S.N.
AUBERT Guy	Physique C.N.R.S
AURIAULT Jean-Louis	Mécanique
AYANT Yves	Physique Approfondie
BARBIER Marie-Jeanne	Electrochimie
BARBIER Jean-Claude	Physique Expérimentale CNRS
BARJON Robert	Physique Nucléaire ISN
BARNOUD Fernand	Boichimie Macromoléculaire Végétale
BARRA Jean-René	Statistiques-Mathématiques Appliquées
BELORISKY Elie	Physique C.E.N.G- D.R.F.
BENZAKEN Claude	Mathématiques Pures
BERNARD Alain	Mathématiques Pures
BERTRANDIAS Françoise	Mathématiques Pures
BERTRANDIAS Jean-Paul	Mathématiques Pures
BILLET Jean	Géographie
BOELHER Jean-Paul	Mécanique
BONNIER Jane Marie	Chimie Générale
BOUCHEZ Robert	Physique Nucléaire ISN
BRAVARD Yves	Géographie
CARLIER Georges	Biologie Végétale
CAUQUIS Georges	Chimie Organique`
CHIBON Pierre	Biologie Animale
COHEN ADDAD Jean-Pierre	Physique
COLIN DE VERDIERE Yves	Mathématiques Pures
CYROT Michel	Physique du Solide
DEBELMAS Jacques	Géologie Générale
DEGRANGE Charles	Zoologie
DELOBEL Claude	Mathématiques Appliquées
DEPORTES Charles	Chimie Minérale
DESRE Pierre	Electrochimie
DOLIQUE Jean-Michel	Physique des Plasmas
DOUCE Rolland	Physiologie Végétale
DUCROS Pierre	Cristallographie
FONTAINE Jean-Marc	Mathématiques Pures
GAGNAIRE Didier	Chimie Physique
GERMAIN Jean-Pierre	Mécanique,
GIRAUD Pierre	Géologie
HICTER Pierre	Chimie
IDELMAN Simon	Physiologie Animale
JANIN Bernard	Géographie
JOLY Jean-René	Mathématiques Pures
KAHANE André, détaché	Physique
KAHANE Josette	Physique
KRAKOWIAK Sacha	Mathématiques Appliquées
KUPKA Yvon	Mathématiques Pures
LAJZEROWICZ Jeanine	Physique
LAJZEROWICZ Joseph	Physique
LAURENT Pierre-Jean	Mathématiques Appliquées
DE LEIRIS Joel	Biologie

LLIBOUTRY Louis	Géophysique
LOISEAUX Jean-Marie	Sciences Nucléaires I.S.N.
MACHE Régis	Physiologie Végétale
MAYNARD Roger	Physique du Solide
MICHEL Robert	Minéralogie et Pétrographie (Géologie)
OMONT Alain	Astrophysique
OZENDA Paul	Botanique (Biologie Végétale)
PAYAN Jean-Jacques	Mathématiques Pures
PEBAY-PEYROULA Jean-Claude	Physique
PERRIAUX Jacques	Géologie
PERRIER Guy	Géophysique
PIERRARD Jean-Marie	Mécanique
PIERRE Jean-Louis	Chimie Organique
RASSAT André	Chimie Systématique
RENARD Michel	Thermodynamique
RINAUDO Marguerite	Chimie CERMAV
ROSSI André	Biologie
SAKAROVITCH Michel	Mathématiques Appliquées
SAXOD Raimard	Biologie Animale
SENGEL Philippe	Biologie Animale
SERGERAERT Francis	Mathématiques Pures
SOUCHIER Bernard	Biologie
SOUTIF Michel	Physique
STUTZ Pierre	Mécanique
VALENTIN Jacques	Physique Nucléaire I.S.N.
VAN CUTSEM Bernard	Mathématiques Appliquées
VIALON Pierre	Géologie

PROFESSEURS de 2ème Classe

ADIBA Michel	Mathématiques Pures
ANTOINE Pierre	Géologie
ARMAND Gilbert	Géographie
BARET Paul	Chimie
BECKER Pierre	Physique
BEGUIN Claude	Chimie Organique
BLANCHI J.Pierre	STAPS
BOITET Christian	Mathématiques Appliquées
BORNAREL Jean	Physique
BRUANDET J.François	Physique
BRUN Gilbert	Biologie
CASTAING Bernard	Physique
CERFF Rudiger	Biologie
CHARDON Michel	Géographie
CHIARAMELLA Yves	Mathématiques Appliquées
COURT Jean	Chimie
DEMAILLY Jean-Pierre	Mathématiques Pures
DENEUVILLE Alain	Physique
DEPASSEL Roger	Mécanique des Fluides
DERRIEN Jacques	Physique
DUFREYNOY Alain	Mathématiques Pures
GASPARD François	Physique
GAUTRON René	Chimie
GENIES Eugène	Chimie
GIDON Maurice	Géologie
GIGNOUX Claude	Sciences Nucléaires
GILLARD Roland	Mathématiques Pures
GIORNI Alain	Sciences Nucléaires
GUIGO Maryse	Géographie
GUMUCHAIN Hervé	Géographie
GUITTON Jacques	Chimie

HERBIN Jacky	Géographie
HERAULT Jeanny	Physique
JARDON Pierre	Chimie
JOSELEAU Jean-Paul	Biochimie
KERCKHOVE Claude	Géologie
LEBRETON Alain	Mathématiques Appliquées
LONGEQUEUE Nicole	Sciences Nucléaires I.S.N.
LUCAS Robert	Physique
LUNA Domingo	Mathématiques Pures
MANDARON Paul	Biologie
MARTINEZ Francis	Mathématiques Appliquées
MASCLE Georges	Géologie
NEMOZ Alain	Thermodynamique CNRS - CRTBT
OUDET Bruno	Mathématiques Appliquées
PELMONT Jean	Biochimie
PERRIN Claude	Sciences Nucléaires I.S.N.
PFISTER Jean-Claude	Physique du Solide
PIBOULE Michel	Géologie
RAYNAUD Hervé	Mathématiques Appliquées
RIEDIMANN Christine	Mathématiques Pures
ROBERT Gilles	Mathématiques Pures
ROBERT Jean-Bernard	Chimie Physique
SARROT-REYNAULD Jean	Géologie
SAYETAT Françoise	Physique
SERVE Denis	Chimie
STOECKEL Frédéric	Physique
SOUTIF Jeanne	Physique
SCHOLL Pierre-Claude	Mathématiques Appliquées
SUBRA Robert	Chimie
VALLADE Marcel	Physique
VIDAL Michel	Chimie Organique
VIVIAN Robert	Géographie
VOTTERO Philippe	Chimie

MEMBRES DU CORPS ENSEIGNANT DE L' IUT 1

PROFESSEURS de 1ere Classe

BUISSON Roger	Physique IUT 1
DODU Jacques	Mécanique Appliquée IUT 1
NEGRE Robert	Génie Civil IUT 1

PROFESSEURS de 2ème classe

BOUTHINON Michel	EEA. IUT 1
CHAMBON René	Génie Mécanique IUT 1
CHEHIKIAN Alain	EEA. IUT 1
CHENAVAS Jean	Physique IUT 1
CHOUTEAU Gérard	Physique IUT 1
CONTE René	Physique IUT 1
GOSSE Jean-Pierre	EEA.IUT 1
GROS Yves	Physique IUT 1
KUHN Gérard, (Détaché)	Physique IUT 1
MAZUER Jean	Physique IUT 1
MICHOULIER Jean	Physique IUT 1
MONLLOR Christian	EEA.IUT 1
NOUGARET Marcel	Automatique IUT 1
PEFFEN René	Métallurgie IUT 1
PERARD Jacques	EEA. IUT 1
PERRAUD Robert	Chimie IUT 1
TERRIEZ Jean-Michel	Génie Mécanique IUT 1
TOUZAIN Philippe	Chimie IUT 1
VINCENDON Marc	Chimie IUT 1

MEMBRES DU CORPS ENSEIGNANT DE MEDECINE

PROFESSEURS CLASSE EXEPTIONNELLE ET 1ère CLASSE

AMBLARD Pierre	Dermatologie	C.H.R.G.
AMBROISE-THOMAS Pierre	Parasitologie	C.H.R.G.
BEAUDOING André	Pédiatrie-Puericulture	C.H.R.G.
BEZEZ Henri	Orthopédie-Traumatologie	Hopital SUD
BONNET Jean-Louis	Ophtalmologie	C.H.R.G.
BOUCHET Yves	Anatomie	Faculté La Merci
	Chirurgie Générale et Digestive	C.H.R.G.
BUTEL Jean	Orthopédie-Traumatologie	C.H.R.G.
CHAMPETIER Jean	Anatomie-Topographique et Appliquée	C.H.R.G.
CHARACHON Robert	O.R.L.	C.H.R.G.
COUDERC Pierre	Anatomie-Pathologique	C.H.R.G.
DELORMAS Pierre	Pneumophtisiologique	C.H.R.G.
DENIS Bernard	Cardiologie	C.H.R.G.
GAVEND Michel	Pharmacologie	Faculté La Merci
HOLLARD Daniel	Hématologie	C.H.R.G.
LATREILLE René	Chirurgie Thoracique et Cardiovasculaire	C.H.R.G.
LE NOC Pierre	Bactériologie-Virologie	C.H.R.G.
MALINAS Yves	Gynécologie et Qbstétrique	C.H.R.G.
MALLION Jean-Michel	Médecine du Travail	C.H.R.G.
MICOUD Max	Clinique Médicale et Maladies Infectieuses	C.H.R.G.
MOURIQUAND Claude	Histologie	Faculté La Merci
PARAMELLE Bernard	Pneumologie	C.H.R.G.
PERRET Jean	Neurologie	C.H.R.G.
RACHAIL Michel	Hépato-Gastro-Entérologie	C.H.R.G.
DE ROUGEMONT Jacques	Neurochirurgie	C.H.R.G.
SARRAZIN Roger	Clinique Chirurgicale	C.H.R.G.
STIEGLITZ Paul	Anestésiologie	C.H.R.G.
TANCHE Maurice	Physiologie	Faculté La Merci
VERAIN André	Biophysique	Faculté La Merci
VIGNAIS Pierre	Biochimie	Faculté La Merci

PROFESSEURS 2ème CLASSE

BACHELOT Yvan	Endocrinologie	C.H.R.G.
BARGE Michel	Neurochirurgie	C.H.R.G.
BENABID Alim Louis	Biophysique	Faculté La Merci
BENSA Jean-Claude	Immunologie	Hopital Sud
BERNARD Pierre	Gynécologie-Obstétrique	C.H.R.G.
BESSARD Germain	Pharmacologie	ABIDJAN
BOLLA Michel	Radiothérapie	C.H.R.G.
BOST Michel	Pédiatrie	C.H.R.G.
BOUCHARLAT Jacques	Psychiatrie Adultes	Hopital Sud
BRAMBILLA Christian	Pneumologie	C.H.R.G.
CHAMBAZ Edmond	Biochimie	C.H.R.G.
CHIROSSEL Jean-Paul	Anatomie-Neurochirurgie	C.H.R.G.
COLOMB Maurice	Immunologie	Hopital Sud
COMET Michel	Biophysique	Faculté La Merci
CONTAMIN Charles	Chirurgie Thoracique et Cardiovasculaire	C.H.R.G.
CORDONNIER Daniel	Néphrologie	C.H.R.G.
COULOMB Max	Radiologie	C.H.R.G.
CROUZET Guy	Radiologie	C.H.R.G.
DEBRU Jean-Luc	Médecine Interne et Toxicologie	C.H.R.G.
DEMONGEOT Jacques	Biostatistiques et Informatique	Faculté La Merci

DUPRE Alain	Chirurgie Générale	C.H.R.G.
DYON Jean-François	Chirurgie Infantile	C.H.R.G.
ETERRADOSSI Jacqueline	Physiologie	Faculté La Merci
FAURE Claude	Anatomie et Organogénèse	C.H.R.G.
FAURE Gilbert	Urologie	C.H.R.G.
FOURNET Jacques	Hépato-Gastro-Entérologie	C.H.R.G.
FRANCO Alain	Médecine Interne	C.H.R.G.
GIRARDET Pierre	Anesthésiologie	C.H.R.G.
GUIDICELLI Henri	Chirurgie Générale et Vasculaire	C.H.R.G.
GUIGNIER Michel	Thérapeutique et Réanimation Médicale	C.H.R.G.
HADJIAN Arthur	Biochimie	Faculté La Merci
HALIMI Serge	Endocrinologie et Maladies Métaboliques	C.H.R.G.
HOSTEIN Jean	Hépato-Gastro-Entérologie	C.H.R.G.
HUGONOT Robert	Médecine Interne	C.H.R.G.
JALBERT Pierre	Histologie-Cytogénétique	C.H.R.G.
JUNIEN-LAVILLAUROY Claude	O.R.L.	C.H.R.G.
KOLODIE Lucien	Hématologie Biologique	C.H.R.G.
LETOUBLON Christian	Chirurgie Générale	C.H.R.G.
MACHECOURT Jacques	Cardiologie et Maladies Vasculaires	C.H.R.G.
MAGNIN Robert	Hygiène	C.H.R.G.
MASSOT Christian	Médecine Interne	C.H.R.G.
MOUILLON Michel	Ophtalmologie	C.H.R.G.
PELLAT Jacques	Neurologie	C.H.R.G.
PHELIP Xavier	Rhumatologie	C.H.R.G.
RACINET Claude	Gynécologie	C.H.R.G.
RAMBAUD Pierre	Pédiatrie	C.H.R.G.
RAPHAEL Bernard	Stomatologie	C.H.R.G.
SCHAERER René	Cancérologie	C.H.R.G.
SEIGNEURIN Jean-Marie	Bactériologie-Virologie	Faculté La Merci
SELE Bernard	Cytogénétique	Faculté La Merci
SOTTO Jean-Jacques	Hématologie	C.H.R.G.
STOEBNER Pierre	Anatomie Pathologique	C.H.R.G.
VROUSOS Constantin	Radiothérapie	C.H.R.G.

MEMBRES DU CORPS ENSEIGNANT PHARMACIE

AGNIUS-DELORD Claudine	Physique	Faculté La Tronche
ALARY Josette	Chimie Analytique	Faculté La Tronche
BERIEL Hélène	Physiologie et Pharmacologie	Faculté La Tronche
BOUCHERLE André	Chimie et Toxicologie	Faculté Meylan
CUSSAC Max	Chimie Therapeutique	Faculté La Tronche
DEMENGE Pierre	Pharmacodynamie	Faculté La Tronche
JEANNIN Charles	Pharmacie Galénique	Faculté Meylan
LATURAZE Jean	Biochimie	Faculté La Tronche
LUU DUC Cuong	Chimie Générale	Faculté La Tronche
MARIOTTE Anne-Marie	Pharmacognosie	Faculté La Tronche
MARZIN Daniel	Toxicologie	Faculté Meylan
RENAUDET Jacqueline	Bactériologie	Faculté La Tronche
ROCHAT Jacques	Hygiéne et Hydrologie	Faculté La Tronche
SEIGLE-MURANDI Françoise	Botanique et Cryptogamie	Faculté Meylan
VERAIN Alice	Pharmacie Galénique	Faculté Meylan

Je tiens à remercier les membres du jury

Claude Delobel, Professeur à l'Université Paris-Sud, qui suit mon travail depuis de longues années et a su m'apporter son appui et ses encouragements. Il me fait aujourd'hui l'honneur de présider le jury de cette thèse,

Christian Boitet, Professeur à l'U.S.T.M.G, qui a accepté de juger mon travail,

Giovanni Coray, Professeur à l'Ecole Polytechnique Fédérale de Lausanne, qui a effectué une lecture scrupuleuse du manuscrit et m'a permis de l'améliorer,

Gilles Kahn, Directeur de Recherche à l'I.N.R.I.A., dont les travaux ont inspiré le sujet de cette thèse et qui a accepté d'en être rapporteur,

Sacha Krakowiak, Professeur à l'U.S.T.M.G., responsable du groupe Génie Logiciel et Système au Laboratoire de Génie Informatique ; c'est dans cet environnement que j'ai pu mener les recherches qui ont abouti à cette thèse.

Je voudrais aussi remercier mes collègues qui, au cours de discussions parfois ardues mais souvent fructueuses, ont apporté leur concours au développement des idées présentées ici. Je dois notamment remercier les membres du projet Tigre, et parmi eux Vania Joloboff, Mauricio Lopez, Hélène Richy et Fernando Velez.

Dès le début, Jacques André a manifesté un intérêt soutenu pour les travaux que je menais, effectuant des expériences avec les prototypes et contribuant largement à les faire connaître. Qu'il trouve ici l'expression de ma reconnaissance.

Enfin et surtout, je voudrais exprimer toute ma gratitude à des compagnons de travail dont la compétence et l'efficacité ont permis la réalisation de beaucoup d'espoirs.

Xavier Rousset de Pina a porté Edimath sur Macintosh, assurant ainsi une large diffusion à ce qui n'était jusque là qu'un prototype de laboratoire.

Irène Vatton a réalisé le Médiateur de Grif et a fortement contribué au mûrissement des concepts présentés ici. Sans elle, Grif n'existerait simplement pas aujourd'hui.

L'un et l'autre ont de plus accepté la tâche ingrate de la lecture des premières versions du manuscrit. Les pages qui suivent doivent beaucoup à leurs critiques.

Je ne voudrais pas oublier Hassan Bédor, qui s'est joint à notre équipe pour prendre en charge les problèmes d'impression de Grif. C'est grâce à lui que les documents traités par l'éditeur peuvent laisser une trace sur le papier.

à ma mère

à mon père

INTRODUCTION

1. Le sujet

Au cours de ces dernières années le domaine du génie logiciel s'est enrichi d'un ensemble de techniques permettant l'édition structurée des programmes. Grâce au développement de ces techniques, on trouve maintenant des environnements de programmation où l'écriture des programmes est grandement facilitée par des outils interactifs intelligents : ils ont une certaine connaissance de la structure que doit respecter un programme et ils utilisent cette connaissance pour guider et assister l'utilisateur dans l'écriture d'un programme syntaxiquement correct. Cette thèse présente la démarche qui a mené à la conception et à la mise en œuvre d'outils analogues pour l'édition de documents de types très variés.

Bien que les programmes puissent être considérés comme des documents d'un type particulier, les outils développés spécifiquement pour la programmation ne peuvent pas s'appliquer efficacement à d'autres types de documents, surtout si on veut traiter aussi bien une simple lettre qu'un formulaire ou un volumineux rapport technique. C'est pour cette raison que nous avons cherché à dégager des concepts plus adaptés et suffisamment généraux pour couvrir une large variété de documents. Ces concepts ont été mis à l'épreuve par la réalisation et l'expérimentation de deux systèmes.

Les principales différences entre documents et programmes résident dans leur structuration et leur aspect graphique. Alors que les programmes sont structurés d'une façon très rigoureuse, parfaitement définie par la syntaxe du langage de programmation, les documents ont une structure plus lâche et très variable d'un document à l'autre. Sur le plan graphique, les différences sont également importantes, puisque l'aspect visuel des programmes a précisément été conçu pour un affichage sur un écran de terminal ou une imprimante, qui, il y a peu, avaient des possibilités graphiques très limitées. Au contraire, les documents auxquels nous ont habitué les typographes sont d'une grande richesse et un système de production de documents doit prendre en compte cette dimension graphique. De plus, les documents contiennent souvent des éléments de différentes natures, tels que tableaux, schémas, formules mathématiques ou chimiques, photographies, qu'on ne rencontre pas dans les programmes.

Ces différences d'importance justifient une étude spécifique qui considère la grande variété structurelle des documents ainsi que leur richesse typographique et qui définisse les concepts et les techniques à mettre en œuvre pour l'édition structurée des documents, tout en utilisant ou en adaptant ce qui a déjà été développé pour les programmes. L'évolution des matériels informatiques, et notamment l'émergence des imprimantes à laser et des postes de travail puissants dotés d'écrans graphiques, vient à point pour appuyer cette démarche, en la rendant techniquement réalisable et économiquement viable à court terme.

2. Plan de la thèse

La thèse est articulée en cinq chapitres. Le premier tente de faire le point sur les techniques et les outils employés dans l'élaboration et la fabrication des documents. Il s'agit aussi bien des outils utilisés dans le domaine des arts graphiques que dans celui de l'informatique, et toute la gamme est considérée, depuis la création des caractères jusqu'à la mise en page finale, en passant par les manipulations des documents complexes.

Le deuxième chapitre présente une première approche du problème à travers une expérience relativement limitée, mais tout à fait significative, qui concerne la manipulation interactive de formules mathématiques. Cette expérience, qui s'appuie sur le développement d'Edimath, a permis de mieux comprendre les problèmes qui se posent dans l'édition d'objets structurés en liaison avec une image de bonne qualité graphique et à travers une interface "naturelle", qui ne transforme pas l'utilisateur en programmeur.

Le troisième chapitre aborde le domaine plus vaste des documents de type quelconque contenant des éléments de toutes natures. Ce chapitre présente essentiellement le modèle de document qui a été retenu pour la construction d'un système interactif manipulant ce genre de documents. Il comprend deux grandes parties, l'une consacrée à la structuration logique des documents et l'autre à leur présentation graphique. Dans chacune des deux parties, un langage spécifique est proposé pour formaliser les structures de documents et les présentations associées.

Le quatrième chapitre décrit le système Grif qui a été réalisé pour mettre en œuvre ce modèle. Les principaux choix de conception sont discutés et les différents constituants du système sont présentés : compilateurs des langages de définition de structure et de présentation, éditeur, gestionnaire d'interface utilisateur, imprimeur. L'ensemble de ces composants constitue un système souple et ouvert, largement paramétrable, permettant d'éditer des documents structurés de tous types à travers une interface utilisateur "conviviale" (fenêtres, menus, désignation à l'écran) et sous une forme graphique de bonne qualité, aussi bien à l'écran que sur le papier.

Le cinquième chapitre conclut en montrant les bénéfices que l'on peut tirer d'un tel système, en le comparant aux principaux systèmes disponibles ou en développement.

CHAPITRE 1

TYPOGRAPHIE ET INFORMATIQUE

La première partie de ce chapitre est consacrée à un rappel historique sur l'évolution des outils de traitement de documents, aussi bien du point de vue du typographe que du point de vue de l'informaticien. Une deuxième partie présente les outils informatiques les plus couramment utilisés en typographie. La troisième partie fait le point sur les outils employés par les informaticiens pour la production des documents. En conclusion, on propose quelques voies de recherche pour harmoniser les différentes approches. Ce sont quelques unes de ces voies qui sont explorées dans les chapitres suivants.

1. UN HISTORIQUE

La typographie et l'informatique sont deux domaines dont l'interpénétration s'est amplifiée au cours des dernières années. Les matériels mis en œuvre actuellement par les typographes et les imprimeurs utilisent largement l'ordinateur et font appel à des logiciels de plus en plus importants. D'un autre côté, tous les ordinateurs, du plus petit au plus gros, sont maintenant dotés de logiciels de production de documents (certes, pas toujours de qualité typographique) et, sur le plan du matériel, les imprimantes permettent désormais d'obtenir des documents similaires à ceux que produisent les professionnels des arts graphiques, ouvrant ainsi la voie à l'"édition électronique" [Gates]. Mais il n'en a pas toujours été ainsi, et l'évolution des deux techniques au cours de ces dernières années permet de comprendre les problèmes qui se posent aujourd'hui aux typographes et aux informaticiens.

1.1. L'évolution du matériel de typographie

Sans remonter à Gutenberg, on peut considérer qu'à l'origine la typographie utilisait une technique stable, qui avait peu évolué au cours des siècles. C'est depuis la fin du XIX^{ème} siècle que des changements radicaux se sont produits, à un rythme de plus en plus rapide [Traband], [Seybold].

1.1.1. La composition chaude

La technique primitive, mise au point par Gutenberg, était celle de l'alliage de plomb. Le typographe disposait de caractères en relief, réalisés dans cet alliage, et il les assemblait les uns à la suite des autres pour former les lignes, qu'il composait lui-même, coupant éventuellement les mots trop longs en fin de ligne. Ces lignes étaient ensuite réunies pour former des colonnes puis des pages. Il est clair qu'avec cette technique la moindre modification du texte composé coûtait très cher, puisqu'elle nécessitait de recomposer à la main une grande partie du texte, et qu'un bon typographe ne composait guère plus de 1500 signes par heure.

Le processus traditionnel de fabrication des caractères en plomb faisait intervenir plusieurs artistes et artisans. Un *dessinateur de caractères* créait d'abord les formes sur papier. Ces dessins étaient utilisés par le *graveur de poinçons*, pour reproduire les formes en relief à l'extrémité de courtes barres d'acier. Les poinçons ainsi produits étaient frappés sur des blocs de cuivre, qui devenaient des moules en creux, les matrices, utilisées par le *fondeur* pour la fabrication des caractères destinés au *typographe*.

Depuis la fin du XIX^{ème} siècle, la méthode de composition manuelle a été progressivement abandonnée, bien qu'elle subsiste encore pour certains travaux particuliers. Le changement est du à l'invention des Linotype et Monotype qui ont complètement changé le mode de travail des typographes, en les dotant d'un clavier.

La Linotype est un appareil entièrement mécanique qui automatise la manipulation des caractères. Dès qu'une touche est frappée au clavier, la matrice du caractère correspondant (un moule en creux) vient se placer à la suite de la matrice du caractère précédemment frappé. Le linotypiste compose ainsi une ligne, jusqu'à ce qu'il juge qu'elle est suffisament pleine. Il actionne alors un levier, qui ajuste les espaces entre mots en introduisant dans ces espaces un ensemble de pièces en forme de coin, assurant ainsi un espacement régulier des mots. L'alliage de plomb fondu est alors versé sur ce moule, constituant ainsi une ligne où les caractères sont en relief.

La Monotype utilise un procédé légèrement différent, puisque le clavier produit un ruban perforé qui est ensuite lu mécaniquement par la machine qui crée les lignes. La composition des lignes se fait en fondant les caractères un par un à partir des matrices et en les assemblant ensuite.

Les lignes produites par les deux types de machine n'ont plus qu'à être mises en page, et aucune intervention manuelle autre que la frappe au clavier n'est nécessaire pour la construction des lignes de texte. Si la vitesse de composition y a gagné (9000 signes par heure), la modification d'un texte composé reste toujours délicate.

1.1.2. La première génération de photocomposeuses

Dans les années 50, la photocomposeuse est venue supplanter les Linotype et Monotype en supprimant l'alliage de plomb et en le remplaçant par la technique photographique. Fin de la composition chaude, début de la composition froide. La machine ne produit plus des lignes de plomb, mais des colonnes de texte sur film photographique. A l'intérieur de l'engin, on trouve une lampe flash, des matrices de caractères sur film négatif, un objectif et un film photographique ; plus de cuivre ni de plomb.

Pour imprimer un caractère, la machine cherche d'abord l'image négative du caractère parmi les matrices. Lorsque l'image du caractère désiré se trouve devant la lampe, celle-ci délivre un éclair qui marque l'image du caractère, à travers l'objectif, sur le film vierge. Un dispositif mécanique et optique se déplace pour que le prochain caractère vienne s'impressioner sur le film à la suite du précédent. Lorsqu'une ligne a ainsi été composée, le film avance pour la ligne suivante. Lorsque tout le texte est composé, le film subit un traitement chimique qui développe et fixe l'image latente marquée par les éclairs.

Cette photocomposeuse, dite de première génération, est directement inspirée des Linotype et Monotype et sa mécanique complexe ne permet pas de composer plus de 30 000 caractères par heure. Elle est rapidement remplacée par une nouvelle génération de machines, qui imposent réellement la technique photographique.

1.1.3. La deuxième génération

Avec la deuxième génération, les matrices sont regroupées sur des disques ou des tambours tournants qui facilitent l'accès aux caractères. Bien que comportant une partie mécanique (avance du film photosensible, rotation du film matrice, déplacement de la tourelle) cette machine fait aussi une place importante à l'ordinateur, qui commande tout son fonctionnement.

L'entrée du texte à composer se fait sur un support habituel aux ordinateurs de l'époque : la bande perforée. Les bandes sont créées sur des postes de saisie, indépendants de la photocomposeuse. L'évolution des périphériques des ordinateurs remplacera ensuite la bande perforée par d'autres supports et notamment les disques, souples ou durs. Les générations se sont ensuite succédées, mais l'architecture est restée la même : des postes de saisie autonomes qui communiquent directement ou par l'intermédiaire d'un support papier ou magnétique avec une machine qui produit une image composée sur un film photosensible.

1.1.4. La troisième génération

La troisième génération marque le début de la typographie numérique [Bigelow 83b]. Elle change le mode de création des images des caractères. Au lieu d'être engendrée par un procédé purement optique, de la même façon qu'on projette une diapositive, l'image d'un caractère est formée électroniquement sur un écran cathodique, à partir d'une matrice de points (ou une autre représentation équivalente) stockée sous forme numérique sur un disque magnétique. L'image obtenue sur le tube cathodique est projetée optiquement, par un objectif, sur le film. L'avantage de cette solution est une plus grande souplesse d'utilisation. Avec les machines de deuxième génération, le nombre de polices de caractères utilisables était limité par la capacité de la tourelle. Avec l'image électronique des caractères, la seule limite est la capacité des disques magnétiques. De plus, la réalisation de polices spéciales, qui était une opération complexe sur les matrices photographiques de la deuxième génération, est devenue plus aisée avec le support magnétique.

Les machines de troisième génération, bien que faisant une place plus importante à l'ordinateur, comportent toujours une partie mécanique non négligeable, essentiellement pour le déplacement de la partie optique, ce qui pénalise les performances.

1.1.5. La quatrième génération

Avec la quatrième génération, la partie optique disparaît : l'image est formée directement sur le film de sortie par un laser. Plus de tube cathodique, plus d'objectif, le seul mouvement mécanique dans la machine est l'avance du film. La suppression de l'objectif implique qu'il n'y a plus de transformation optique des caractères. Dans les machines des générations précédentes, on utilise l'objectif pour produire des caractères de différentes tailles, simplement par variation de la focale. Cela n'est plus possible avec la quatrième génération, et il faut soit stocker un plus grand nombre de caractères numérisés, soit effectuer des transformations géométriques sur ces caractères numérisés.

Un dernier perfectionnement, réalisé dans la quatrième génération, est dû à des lasers plus puissants et des logiciels plus complets. Au lieu d'impressionner un film photographique, le laser grave directement une plaque d'aluminium, qui est ensuite utilisable immédiatement sur une rotative. Cela suppose que la mise en page est également effectuée par le système.

Les postes de saisie ont également évolué depuis la première génération de photocomposeuses. Les premiers postes comportaient un grand clavier, très proche de celui des linotypes, avec 300 ou 400 touches et ne produisaient que du ruban perforé. Les

opérateurs, venant des linotypes et habitués à ne pas voir immédiatement le résultat de leur travail, tapaient en aveugle.

Maintenant, le poste de saisie d'une photocomposeuse est très proche d'un terminal d'ordinateur. C'est même souvent un terminal connecté à un ordinateur puissant qui gère également la photocomposeuse. Mais si ces terminaux simples permettent la saisie du texte, d'autres appareils, beaucoup plus sophistiqués, sont utilisés pour effectuer la mise en page, et particulièrement dans la presse : une page de journal est plus complexe qu'une page de roman et il faut la composer plus rapidement. Ces postes s'apparentent aux postes de travail modernes des informaticiens, avec notamment un grand écran graphique et une souris ou un appareil de désignation équivalent.

1.2. L'évolution des outils informatiques.

Ainsi, l'ordinateur a pris progressivement une place prépondérante dans les systèmes de photocomposition et ce qu'on voit aujourd'hui dans un atelier de photocompostion moderne ressemble tout à fait à un centre de calcul. Le numéro de la prochaine génération de machines attendue par les informaticiens et les typographes est le même, et ce n'est pas le seul point commun. On peut effectivement comparer l'évolution du matériel informatique à celle du matériel typographique.

1.2.1. Les imprimantes et les écrans

C'est sans conteste l'évolution des imprimantes et des écrans qui a attiré les informaticiens vers la typographie. Les télétypes et les imprimantes à chaîne ou à tambour produisaient des résultats de trop mauvaise qualité pour envisager de les utiliser dans le traitement d'autres documents que les programmes et éventuellement leur documentation.

Il a fallu attendre l'arrivée des imprimantes à boule ou à rosace pour que les sorties d'ordinateur deviennent comparables à des documents dactylographiés. Bien que n'approchant pas encore la qualité typographique, les nouvelles possibilités offertes par ces machines ont stimulé le développement d'outils de traitement de texte. Le progrès principal était la qualité de frappe, mais celle-ci était obtenue au détriment de la vitesse (quelques dizaines de caractères par seconde seulement). De plus le choix des caractères était très limité (une centaine de caractères simultanément).

Les imprimantes matricielles à aiguilles sont apparues ensuite et ont offert de nouvelles possibilités en multipliant les jeux de caractères disponibles et en proposant certains traitements graphiques. Leur qualité de frappe est encore insuffisante pour

produire des documents de bonne qualité, mais elles sont plus riches de possibilités et souvent plus rapides que les précédentes.

Ce sont les imprimantes "non-impact" (à laser, à diodes, à déposition d'ions, magnéto-graphiques, etc...) qui ont enfin permis aux systèmes informatiques de sortir des documents semblables, à la résolution près, à ceux des typographes. Ces appareils offrent à la fois une bonne qualité d'impression, un débit important et des fonctionalités riches : nombreux jeux de caractères, de toutes tailles, positionnement libre de ces caractères dans la page, un mode graphique, etc... C'est seulement avec l'apparition de ces imprimantes qu'on a pu passer du simple traitement de texte à la manipulation de documents complexes.

Les outils modernes de production de documents ont également été rendus possibles par les progrès réalisés dans le domaine des terminaux et des techniques d'affichage, qui ont conduit jusqu'aux stations de travail d'aujourd'hui. Avec les premiers terminaux, qui écrivaient sur papier, le travail d'édition des documents était trop complexe pour qu'on puisse mettre un système de traitement de texte entre toutes les mains. Les terminaux alpha-numériques à écran ont fait tomber cette barrière, en permettant une édition réellement interactive. Il est ainsi devenu possible d'afficher un texte de plusieurs lignes très rapidement et surtout de le modifier directement sur l'écran. Il reste cependant l'inconvénient de la structure rigide de la grille d'affichage (les possibilités de positionnement de caractères sont très limitées) et celui de la variété des caractères affichables (moins de cent).

Un autre inconvénient de ces appareils est qu'ils n'affichent que du texte. Les terminaux graphiques disponibles au même moment n'avaient aucun de ces défauts, mais leur usage dans la manipulation des textes était limité par leur prix et aussi par leur mode de fonctionnement, essentiellement basé sur le tracé de vecteurs, qui se prête mal au dessin de caractères.

C'est la technique des matrices de points (bit-map) qui a ouvert de nouvelles possibilités pour l'affichage de documents sur écran. Elle permet tout à la fois une grande vitesse d'affichage, une totale liberté de placement des éléments d'image, un choix infini de caractères (forme, taille, style), et des capacités graphiques variées, qui vont du tracé de vecteur jusqu'à l'affichage de photographies tramées. De plus cette technique est généralement mise en œuvre sur des machines qui offrent une bonne puissance de traitement et une capacité mémoire importante. L'écran est de grande taille, permettant d'afficher au moins une page de texte, et il est accompagné d'une souris, pour la désignation directe sur l'image. Ce type de poste de travail a été créé avec la machine Alto [Thacker] et s'est depuis largement répandu.

Les évolutions simultanées des techniques d'impression et d'affichage sur écran ont permis de disposer au même moment d'appareils de caractéristiques voisines, se mariant bien dans un système de production de documents. Les écrans alpha-numériques correspondent aux imprimantes à rosace, tout comme les écrans à matrices de points correspondent aux imprimantes à laser.

1.2.2. L'évolution des logiciels de production de documents

Les matériels disponibles ont largement déterminé l'évolution des systèmes de production de documents. Ceux-ci ont démarré timidement dans les années 60, d'abord comme de simples prolongements des outils d'édition de programmes, avec les formateurs [Furuta 82]. Ils ont ensuite pris une grande extension et ils constituent maintenant toute une classe d'outils très variés [André 82, van Vliet].

Les premiers systèmes étaient des formateurs de bas niveau, qui traitaient un texte saisi à l'aide du même éditeur que celui utilisé pour l'écriture des programmes, quand le texte n'était pas simplement sur carte perforées. Le rôle de ces formateurs était très simple. Il consistait essentiellement à construire des lignes d'égale longueur et à former des pages, à partir d'un texte "au kilomètre". Quelques commandes pouvaient être glissées dans le texte pour contrôler cette mise en page. Il s'agissait essentiellement de commandes élémentaires, très proches de celle traitées par l'imprimante (espaces, sauts de lignes, sauts de page...). Les possibilités de formatage étaient limitées par les capacités des imprimantes : peu de caractères disponibles, chasse fixe et positionnement grossier.

Le langage dans lequel s'exprimaient les commandes était peu formalisé et de bas niveau, comparable à un langage d'assemblage. Comme les langages de programmation, ces langages ont évolué. Ils ont d'abord été dotés de macro-instructions, permettant ainsi d'adapter le formateur et d'offrir à l'utilisateur des commandes d'un niveau plus élevé. Ces commandes se sont également enrichies pour permettre de tirer parti des possibilités de nouvelles imprimantes, voire même de photocomposeuses, comme dans Troff [Kernighan 78].

Les formateurs ont également été enrichis pour traiter des éléments non-textuels. L'ensemble des outils de formatage disponibles sous UNIX [Ritchie] constitue un bon exemple de cette évolution. Un ensemble de préprocesseurs permet d'inclure différents types d'objets dans les documents destinés à Troff ou Nroff : Tbl [Lesk] traite les tableaux, Eqn [Kernighan 75] traite les formules, Pic [Kernighan 82] traite les schémas. D'autres outils encore assistent l'écrivain en prenant en charge les références bibliographiques ou en détectant les fautes d'orthographe ou même de style [Cherry].

Une autre approche a été développée à la fin des années 70, notamment par B. Reid [Reid 80], en considérant le langage des formateurs comme un langage de haut niveau qui décrit le document à produire en termes logiques et non plus en fonction de la présentation souhaitée. Le rôle du formateur est devenu plus important, puisqu'il doit, à partir de la description logique d'un document, décider de sa mise en page avant de produire son image sur une imprimante. A la même époque, l'arrivée des imprimantes à laser a également poussé les formateurs vers une plus grande qualité typographique.

Le complément indispensable d'un formateur est un éditeur [Meyrowitz] qui assure la saisie et la mise à jour de la description du document qui doit être soumise au formateur. D'abord cantonnés dans ce rôle, certains éditeurs ont évolué en intégrant des fonctions de formatage, pour devenir des éditeurs-formateurs, qui produisent un document imprimable directement, puisqu'il est déjà mis en page. Mais si ces systèmes n'ont pas supplanté les formateurs, c'est que les fonctions de formatage correspondent à celles qu'on trouvait sur les tout premiers formateurs, c'est à dire des fonctions de bas niveau. Néanmoins, ces éditeurs-formateurs ont connu un succès important, d'abord sur des machines spécialisées de traitement de texte, puis comme logiciel d'application sur les micro-ordinateurs.

La dernière évolution de ce type de système est celle qui utilise des écrans à matrice de points et des imprimantes à laser pour afficher sur l'écran une image strictement conforme à celle qui sera imprimée. C'est l'approche "What You See Is What You Get" (WYSIWYG), d'abord développée sur l'Alto avec Bravo [Lampson] puis reprise dans des produits comme le Star [Harslem] ou le Macintosh [Williams]. Elle permet de tirer parti de toutes les possibilités du matériel (nombreux styles et tailles de caractères, graphique intégré) et offre une interface utilisateur évoluée.

2. LES OUTILS INFORMATIQUES DE LA TYPOGRAPHIE

La technique informatique n'a pas seulement servi à construire des systèmes de production de documents comme les formateurs ou les éditeurs-formateurs qui ont été mis entre les mains des informaticiens et de nombreux utilisateurs des ordinateurs. Elle a aussi permis le développement d'outils spécifiquement utilisés par les typographes, notamment pour traiter les caractères et composer des pages.

2.1. Le traitement des caractères

Le caractère est la matière première du typographe. Dans les photocomposeuses modernes il est stocké et traité uniquement sous forme électronique. Il n'est plus matérialisé ni par le plomb ni par une image photographique négative. Mais avant d'en arriver là, les caractères ont subi un longue évolution, aussi bien dans leur forme que dans les moyens de les créer et de les représenter.

Les caractères d'imprimerie que nous lisons aujourd'hui sont issus d'origines différentes [Bigelow 83a]. Les capitales sont pratiquement identiques aux lettres que les romains gravaient dans la pierre de leurs monuments. Le trait suivi par le burin du graveur formait les lettres qui, de ce fait, étaient définies par leur contour, leur "glyptique". Elles revêtaient une certaine solennité et devaient être lisibles par tous. Leur forme était donc très rigoureusement respectée et elle est parvenue jusqu'à nous pratiquement intacte.

Au contraire, nos lettres minuscules (les bas de casse des typographes) résultent de l'évolution des caractères manuscrits, tracés rapidement à la plume sur le parchemin ou le papier. Destinés à des lettrés et à un usage privé, ces textes, le plus souvent de nature juridique ou comptable, pouvaient tolérer quelques distortions dans leur calligraphie. Là, c'est le mouvement de la main, le "ductus", qui définit la structure géométrique de la lettre. Le ductus a évolué au cours des siècles avec les caractéristiques des outils d'écriture, ce qui explique que les textes manuscrits de l'époque romaine soient très difficile à lire aujourd'hui, au contraire des inscriptions monumentales. Cette évolution a abouti à la minuscule caroline qui s'est développée dans les monastères de la période carolingienne. C'est le véritable ancêtre de nos caractères bas de casse. Avec l'invention de l'imprimerie, sa forme s'est figée et a alors été définie, pour les graveurs de poinçons, par son contour. Du fait de la technique des caractères en plomb, ce sont ensuite des transformations de la glyptique, et non plus du ductus, qui sont intervenues. Mais les évolutions ont été moins importantes depuis les dessinateurs de caractères de la Renaissance qui ont conservé la forme donnée par la main du copiste.

La numérisation des caractères est un nouveau facteur qui influence leur forme. De même que les précédentes évolutions techniques, celle-ci commence par reproduire le plus fidèlement possible les caractères existants, de façon à respecter les habitudes de l'œil du lecteur. Progressivement, de nouveaux caractères sont créés, qui prennent mieux en compte les possibilités et les limites du support numérique [Bigelow 86].

L'origine des formes des caractères n'influe pas seulement sur la création de nouvelles polices, mais aussi sur la conception des outils qui aident à créer et à numériser les caractères. Deux systèmes sont particulièrement intéressants de ce point de vue, Metafont et Ikarus.

L'utilisation brutale d'un scanner pour numériser des caractères dessinés sur papier est insuffisante : on observe de nombreuses anomalies sur les contours et la représentation sous la forme d'une simple matrice de points rend les transformations et le traitement des caractères pratiquement impossibles. Ikarus [Karov] résoud ces problèmes. C'est avant tout un système pour la numérisation de caractères existant. Pour les caractères électroniques, il joue le rôle du graveur de poinçons qui transpose sur un autre support une forme dessinée sur le papier. Il considère le contour des lettres et s'intéresse donc essentiellement à la glyptique des caractères. L'opérateur entre, à l'aide d'une tablette à digitaliser, un certain nombre de points caractéristiques du contour de la lettre. Ces points sont classés en quatre catégories : point de départ (premier point du caractère), extrémité d'une partie rectiligne, point d'une partie courbe, point de tangence. Selon le style de la police, entre 20 et 60 points sont nécessaires pour représenter complètement un caractère. A partir de cette définition géométrique, le contour complet est calculé par des fonctions splines et différentes représentations du caractère peuvent être produites à différentes résolutions. Le caractère peut être édité (déplacement ou ajout de points caractéristiques) ou transformé géométriquement (agrandissement, réduction, inclinaison, expansion, contour, ombrage, interpolation, etc...). Ikarus est un outil largement utilisé dans l'industrie typographique.

Metafont [Knuth 85] suit une approche très différente. Il prend en charge à la fois la création des caractères et leur numérisation. A la différence de Ikarus, il est conçu pour la création de nouveaux caractères, plus que pour la simple numérisation de caractères existants. Il permet même de créer des familles de polices, et non pas seulement des caractères isolés : l'objectif est d'obtenir "des descriptions de haut niveau qui transcendent toutes les polices décrites individuellement". Ces descriptions sont en fait des programmes de nature déclarative, écrits dans le langage propre à Metafont, qui indiquent à la machine comment tracer des caractères. Metafont considère essentiellement le ductus ; chaque caractère est défini par le parcours d'une plume virtuelle dont plusieurs paramètres peuvent varier (forme, taille, inclinaison). Le parcours de la plume est également paramétré. Si on choisit correctement les paramètres qui agissent sur l'ensemble d'une police on peut ensuite faire varier de façon homogène certaines parties des caractères (par exemple des empattements ou des courbures) pour mettre au point globalement les formes des caractères ; on peut aussi opérer des transformations comme le graissage, l'inclinaison ou l'expansion, pour engendrer d'autres polices de la même famille.

Metafont est un système puissant, qui permet au créateur de caractères d'exprimer les principes d'un alphabet d'une façon rigoureuse. C'est aussi un assistant efficace qui redessine les caractères dès que c'est nécessaire, après que quelques modifications ont été effectuées. Mais c'est surtout un outil destinés aux professionnels des arts graphiques qui ont de plus une compétence en programmation. L'utilisation d'un tel outil n'est toujours facile pour le profane [André 85a].

Une autre approche est proposée par [Flowers], qui s'inspire des techniques de CAO. Le système LIP est complètement interactif, à la différence des deux précédents, et permet au créateur de caractères de travailler directement sur la forme graphique des caractères affichés à l'écran. Comme dans Ikarus, un caractère est représenté par son contour, lui-même défini par des points caractéristiques. La méthode de saisie des caractères est la même, mais les possibilités de modification sont plus importantes, du fait de l'interactivité : on peut déplacer les points caractéristiques à l'écran, copier certaines parties de caractères (empattements par exemple) et les réutiliser dans d'autres caractères, avec transformation (rotation, symétries, changement d'échelle...). Le système offre également les moyens d'assurer l'homogénéité en contrôlant certains alignements ou certaines dimensions sur un ensemble de caractères.

Les trois systèmes que nous venons de voir sont destinés à des professionnels des arts graphiques qui ont une bonne connaissance de la lettre. Cependant le rapide développement des postes de travail à écran graphique et des imprimantes à laser a poussé les informaticiens à créer des outils pour le traitement des caractères numérisés. On trouve maintenant sur toutes les machines graphiques des *éditeurs de polices* destinés à produire des caractères représentés sous forme de matrices de points à faible résolution (moins de 100 points par pouce). Comparés aux précédents, ces outils sont souvent très primaires. Ils sont essentiellement conçus pour poser des points blancs et noirs dans une grille rectangulaire représentant la matrice d'un caractère. En plus de cette fonction de base, ils sont parfois dotés de quelques possibilités d'édition : décalages, rotations, symétries, superpositions de caractères ou de fragments de caractères. Certains peuvent effectuer quelques tracés géométriques simples : segments de droite, rectangles, cercles.

Si ces éditeurs de polices connaissent quelques rudiments de géométrie, ils n'ont aucune expertise typographique. Or ils sont le plus souvent entre les mains d'utilisateurs ignorant tout de l'art du dessin des caractères et des contraintes de la numérisation, particulièrement importantes avec les faibles résolutions des machines les plus courantes. C'est probablement ce qui explique la piètre qualité des textes affichés sur les écrans. Les faiblesses de l'outil devraient être compensées par l'expertise de l'utilisateur, mais malheureusement l'expertise des typographes n'est pas encore passée dans le domaine de l'informatique, et c'est probablement au niveau des caractères que se posent les problèmes les plus nombreux. On verra plus loin que des outils existent qui résolvent un certain nombre d'autres difficultés typographiques, sans demander aux utilisateurs d'être eux-mêmes des typographes. Pour la création des caractères, il faut soit s'en remettre aux experts, soit suivre les quelques conseils qu'ils peuvent donner [Bigelow 85].

Certains dessinateurs de caractères commencent à s'intéresser aux contraintes spécifiques des écrans et imprimantes à basse résolution. En effet, un écran courant affiche entre 60 et 100 points par pouce, une imprimante laser environ 300 points. Les

photocomposeuses, elles, ont une résolution comprise entre 1200 et 4000 points par pouce. Cette différence change radicalement le problème de la numérisation des caractères, mais aussi celui de leur dessin. La famille de polices Lucida [Bigelow 86] est l'une des premières qui ait été conçue pour cette nouvelle génération d'appareils.

Le traitement des caractères ne se limite pas à la saisie ou à la création. La forme sous laquelle ils sont stockés dans les systèmes de photocomposition est aussi importante [Hégron]. C'est elle qui détermine l'espace nécessaire sur les disques, ainsi que les traitements que peut effectuer le système : il est par exemple difficile d'opérer des transformations géométriques sur les caractères à partir d'une représentation sous forme de matrice de points. De plus les matrices de points deviennent très volumineuses dès que la résolution ou le corps augmente.

Une variante de cette représentation a été proposée par [Warnock 80]. Elle atténue les effets de la numérisation sur les contours par un sur-échantillonnage qui produit des points gris. Si elle améliore la qualité de l'image pour une définition donnée, cette méthode augmente le volume de stockage nécessaire. Une représentation plus compacte, et mieux adaptée au balayage des tubes cathodiques ou des lasers, est celle des longueurs de balayage : chaque ligne de la matrice de points est représentée par une suite de segments blancs et noirs de longueur donnée : on ne code plus chaque point par un bit, mais chaque suite de points de même couleur par un entier, la longueur de la suite. Enfin une représentation du contour par des courbes a l'avantage d'occuper peu de place, quelle que soit le corps ou la résolution, et de permettre des transformations, mais les calculs pour produire le signal à envoyer au dispositif de marquage sont importants.

2.2. La composition

Pour produire un document imprimé, une fois qu'on dispose de caractères, il faut ensuite les assembler pour constituer des mots. Cette opération ne peut être réalisée que si des informations complémentaires sont associées aux formes des caractères, pour indiquer les alignements (*ligne de base*) et les distances (*approches* et *talus*) entre caractères voisins. En bonne typographie, ces distances ne sont pas liées à chaque caractère individuellement, mais varient selon les caractères voisins. Ainsi, un 'T' majuscule se rapproche d'avantage du caractère précédent si celui-ci est un 'A' majuscule que si c'est un autre 'T'. Ce phénomème est appelé *crénage*.

Avec certaines polices (mais pas celle qui est utilisée ici), il existe des combinaisons de lettres (ff, fl, fi, ffl, ffi) dont le dessin a été conçu globalement et diffère du graphisme produit par la succession des lettres élémentaires ; par exemple, le point du 'i' vient se fondre dans l'extrémité de la boucle du 'f' qui le précède. Une telle *ligature* ne peut pas être

traitée comme un caractère unique puisqu'elle est susceptible d'être éclatée par une coupure de mot en bout de ligne.

Les règles d'assemblage des caractères se compliquent encore avec les traditions typographiques de chaque langue. La position des signes de ponctuation, en particulier, n'est pas la même en français et en anglais. La typographie française laisse un espace avant les signes ';' ':' '?' et '!', alors que la typographie anglaise place ces mêmes signes immédiatement à la fin du mot précédent.

Toutes ces règles d'assemblage des caractères sont généralement prises en compte dans les systèmes typographiques, qui travaillent avec des tables de crénage, recherchent les ligatures et placent les signes de ponctuation. Avec les mots ainsi constitués, ils composent des lignes. Dans cette tâche aussi, l'art du typographe a été remplacé par des algorithmes [Achugbue]. Il s'agit de déterminer l'endroit, entre deux mots, où le texte doit être coupé de façon à obtenir des lignes de longueur aussi peu variable que possible. Le problème est compliqué par le fait que les espaces qui séparent les mots ne sont pas tous des points de coupure possibles. Ainsi, on évite de changer de ligne entre un mot et le signe de ponctuation qui le suit, entre l'initiale d'un prénom et le nom propre qui suit (J. Dupont), entre les chiffres d'un même nombre (1 000 000), entre une somme en chiffres et le symbole de la monnaie (100 F), etc...

Une fois les lignes obtenues, elles peuvent être justifiées par compression ou extension des espaces entre les mots. Sur des justifications étroites, comme les colonnes des journaux, il peut être nécessaire de faire varier les espaces entre caractères pour éviter les blancs trop importants entre les mots.

Le problème du découpage en lignes est lié à celui de la coupure des mots, puisqu'une meilleure répartition des espaces peut être obtenue lorsque les mots de bout de ligne sont coupés. Selon les langues, le problème se pose différemment. En français, il existe des règles générales [Grevisse] qui permettent une approche algorithmique du problème [Buckle]. Bien que cette approche ne soit pas parfaite [Désarménien], elle a l'avantage de la simplicité. Au contraire, en anglais, les règles sont moins bien formalisées et la seule référence est le dictionnaire (un bon dictionnaire anglais indique pour chaque mot les points où la coupure est autorisée). La consultation d'un dictionnaire complet est une méthode lourde et couteuse, aussi des techniques spécifiques ont-elles été développées pour structurer et compacter les dictionnaires de coupure, de façon à les rendre utilisables par des algorithmes efficaces [Liang]. Cette technique a également été adaptée à la langue française [Désarménien].

Une bonne technique de coupure de mots doit satisfaire deux critères. Elle ne doit pas commettre d'erreur et donc ne couper les mots que là où c'est autorisé par l'usage de la langue. Elle doit aussi être complète et trouver dans les mots candidats à la coupure toutes

les coupures possibles, de façon à ajuster au mieux la largeur des espaces. Les coupures de mots améliorent la construction des lignes, mais elles la compliquent également, puisqu'un bon typographe évite les trop nombreuses coupures de mots et particulièrement dans les lignes successives et en bas de page.

Le problème de la construction des lignes est lié à celui de la construction des paragraphes. Une approche globale du problème est présentée par [Plass]. Au lieu de prendre les décisions de coupure ligne par ligne, la méthode proposée considère l'ensemble du paragraphe, en tenant compte de l'influence sur une ligne du texte des lignes suivantes. L'algorithme détermine les points de coupure optimum tout en évitant les retours en arrière grâce à l'utilisation des techniques de la programmation dynamique. Ce sont ces mêmes techniques qui sont mises en jeu dans l'algorithme proposé par [Achugbue].

2.3. La mise en page

L'assemblage des caractères en mots, en lignes et en paragraphes donne de longues colonnes de texte, qu'il faut encore couper et organiser en pages, en insérant des éléments supplémentaires, comme les figures, les illustrations, les tableaux, les notes, etc... Ce travail de montage, traditionellement effectué avec des ciseaux et de la colle, peut également être automatisé, ou au moins assisté par ordinateur.

Dans la presse notamment, il existe des postes de travail graphiques, où la mise en page se fait de façon interactive sur un écran, simulant le travail fait jusque là sur une table de montage [Sundblad]. Ces systèmes permettent de faire la mise en page, à partir d'éléments préexistants, comme des images numérisées (photographies, dessins), et du texte saisi préalablement et mis en colonnes. Ils permettent d'ajouter des éléments graphiques (filets, cadres, tableaux), mais leur principale fonction est de placer et de déplacer tous les éléments dans la page, sous le contrôle de l'opérateur. Certains de ces systèmes sont aussi capables de traiter les images [Blomberg] : agrandissements, réductions, rotations, modifications du contraste ou de la densité, retouches.

Ces outils, d'abord développés pour les professionnels des arts graphiques, utilisent des matériels puissants et aux fonctionalités graphiques développées. Mais on trouve aussi des logiciels comparables, au moins pour ce qui est des fonctionalités, sur des machines beaucoup plus simples, et qui sont maintenant destinés à un usage de bureau. On peut citer parmi ces logiciels Page Maker, Ventura Publisher ou Ready Set Go 3.

3. LES SYSTEMES DE PRODUCTION DE DOCUMENTS : ETAT DE L'ART

Après avoir examiné la nature du travail typographique et les outils dont disposent les professionnels, nous présentons ici les systèmes qui ont été développés par les informaticiens dans le domaine de la production des documents, et qui sont le résultat de l'évolution retracée au début de ce chapitre.

Nous essayons d'abord de classer et de décrire les différents outils selon leurs fonctions, puis nous présentons d'une façon plus détaillée deux systèmes représentatifs : Scribe et Interleaf. Enfin, nous faisons le point sur l'état de la normalisation concernant les documents.

3.1. Une taxinomie des systèmes de production de documents

L'ensemble des systèmes de traitement de documents peut être partagé en cinq catégories : les éditeurs et systèmes de traitement de texte, les formateurs, les éditeurs syntaxiques, les outils spécialisés et les interfaces utilisateur.[1]

3.1.1. Les éditeurs et systèmes de traitement de texte

Employés seuls, les éditeurs d'usage général [Meyrowitz] ne peuvent produire que des documents d'aspect relativement grossier. Ils sont le plus souvent conçus pour la manipulation de programmes et, pour d'autres types de documents, ils ne peuvent guère servir qu'à préparer un travail qui sera ensuite soumis à un formateur. Il y a d'ailleurs une certaine parenté entre la programmation et l'utilisation des formateurs. La classe des éditeurs est bien représentée par Emacs [Stallman], qui regroupe la plupart des fonctions que l'on trouve habituellement dans ce type d'outil. Emacs a inspiré le développement de nombreux autres éditeurs et on ne compte plus les imitations plus ou moins fidèles.

Les systèmes de traitement de texte constituent une famille d'outils mieux adaptés au traitement des documents. Ils sont en fait, dans leur principe, dérivés des éditeurs, dont ils reprennent les fonctions de base. Ils ajoutent des fonctions complémentaires permettant une présentation plus élaborée des documents. Ils autorisent le centrage et la justification, ils permettent d'affecter à certaines partie du texte des attributs "typographiques" tels que italique, gras ou souligné. Ces systèmes prennent également en charge l'impression des documents. Ils effectuent la pagination et, lors de l'impression, ils peuvent aussi jouer sur certains paramètres comme l'interligne ou les marges.

---

[1] On pourrait ajouter une catégorie supplémentaire, celle des systèmes de génération de notes de bas de pages. Voir notamment [Nedginn].

Conçus pour être mis entre les mains de tout le monde, les systèmes de traitement de texte sont souvent simples d'utilisation ; mais leurs fonctionnalités sont malgré tout limitées, et ils laissent à leur utilisateur une partie importante du travail dès que le document atteint une certaine complexité. C'est l'utilisateur qui doit numéroter les chapitres, les sections, les figures, etc... C'est lui qui doit gérer les renvois ("voir figure *n*"), établir la table des matières, la table des figures, les index. Il doit surtout mettre à jour tous ces éléments dès qu'il apporte des changements au document. C'est encore lui qui doit contrôler l'uniformité de la présentation et qui doit vérifier que la construction des paragraphes et des pages est correcte (éviter, par exemple, qu'un paragraphe commence sur la dernière ligne d'une page, ou se termine sur la première ligne d'une autre page). Bref, ce type de système est plutôt adapté au traitement des documents simples.

Pour les documents fortement structurés, d'autres systèmes sont proposés, dans la classe des "processeurs d'idées". Il s'agit plutôt d'outils destinés à aider l'organisation des idées qui seront ensuite exprimées dans un document. Ils sont utilisés dans la première phase d'élaboration du document. ThinkTank [Hershey] a été le premier système de cette catégorie. Il permet essentiellement de manipuler une table des matières, où la hiérarchie des sections est clairement indiquée sur l'écran, par des retraits. De nombreuses opérations sont possibles sur les sections, à tous les niveaux, comme des créations, suppressions, changements de niveau, déplacements, permutations ou classements alphabétiques. L'utilisateur contrôle la granularité de l'image qui lui est présentée : il peut faire afficher uniquement les titres du niveau le plus élevé et ainsi voir l'ensemble du document d'une façon synthétique ; il peut aussi faire afficher les titres des *n* premiers niveaux d'une section donnée s'il veut entrer dans le détail de cette section. Après chaque titre de section, il peut insérer un texte libre qui n'est affiché que sur commande. Enfin, l'impression des documents est assurée par le système, avec une possibilité d'ajouter les numéros des sections.

Les processeurs d'idées résolvent certains problèmes posés par les systèmes de traitement de texte, mais ils n'offrent pas toutes les possibilités d'édition et de présentation du texte lui-même. Ils s'intéressent plus à la structure du document qu'à son contenu. Il s'agit en fait d'outils complémentaires, et non concurents, des traitements de texte.

Mentionnons enfin une évolution récente des systèmes interactifs, qui a abouti aux systèmes de production de documents techniques. Bien qu'ils ne couvrent pas la totalité du domaine complexe de la documentation technique, ces systèmes en abordent plusieurs aspects, et notamment l'insertion et le traitement d'images et de schémas dans les documents. On trouvera plus loin une description détaillée du premier représentant de cette classe qui soit apparu sur le marché, Interleaf [Morris 85].

3.1.2. Les formateurs

Tous les systèmes précédents sont interactifs : l'utilisateur voit sur l'écran une image du document qu'il traite et chaque modification qu'il apporte au document se reflète immédiatement sur l'écran. Il peut ainsi réagir rapidement s'il n'obtient pas le résultat attendu. Les formateurs, eux, sont essentiellement des outils non-interactifs. On leur soumet un travail décrit dans un fichier, ils effectuent ce travail et délivrent le résultat global après un certain délai. L'utilisateur n'a aucun moyen d'intervenir sur le déroulement du traitement. Si le résultat ne le satisfait pas, il doit modifier, à l'aide d'un éditeur, le fichier décrivant ce qu'il veut obtenir et relancer l'exécution de l'ensemble.

Bien que ce mode de travail puisse paraître lourd, cette lourdeur est compensée par une grande puissance de traitement. La plupart des formateurs effectuent des opérations inconnues dans les systèmes de traitement de texte, et notamment, les numérotations, les constitutions de tables ou d'index, ou la gestion des renvois. Ils se distinguent également par une plus grande qualité de la typographie, particulièrement en ce qui concerne la coupure des mots, les crénages, les ligatures, la construction des lignes et des paragraphes.

On peut distinguer deux classes parmi les formateurs récents : les formateurs de haut niveau et les formateurs de bas niveau offrant un mécanisme de macro-instructions. Les premiers sont notamment représentés par Scribe [Reid 83], dont les caractéristiques originales méritent d'être présentées en détail (voir section 3.2). Les seconds regroupent des systèmes comme TEX ou Troff.

Les formateurs de haut niveau travaillent à partir d'une description logique des documents. L'utilisateur n'est pas contraint de spécifier les détails de la présentation qu'il souhaite obtenir. Il indique seulement le type des différentes parties du document (titre, résumé, section,...) et le formateur se charge de la présentation, ce qui permet notamment d'obtenir une grande homogénéité dans l'aspect des documents imprimés, puisque les éléments de même type sont présentés de la même façon. Les traitements mentionnés plus haut se fondent également sur cette description logique.

Les formateurs de bas niveau permettent d'inclure dans la description du document des commandes qui produisent des changements de police ou des espacements, des modifications des marges, des justifications différentes, etc..., en fait les mêmes commandes que celles qui sont proposées dans les systèmes de traitement de texte. Mais ces formateurs offrent en plus un moyen de spécifier des commandes de plus haut niveau, analogues à celles des formateurs de la catégorie précédente. Cela se fait par des macro-commandes qui peuvent être définies dans le fichier contenant le document lui-même ou dans des bibliothèques. Avec les bibliothèques, les macro-commandes sont utilisables dans tous les documents et le formateur se rapproche alors de la catégorie précédente. LaTEX [Lamport] est une illustration de cette démarche, puisque son ambition

est d'offrir les fonctionalités de Scribe en s'appuyant sur TEX. De même la bibliothèque -ms de Troff permet de décrire les documents à un plus haut niveau que ne l'autorise Troff seul.

Malgré les traitements qu'ils effectuent et la qualité de leur typographie, les formateurs se voient souvent reprocher leur manque d'interactivité. Aussi des outils complémentaires ont-ils été développés pour combler, au moins partiellement, cette lacune. Ce sont les systèmes d'épreuvage, qui affichent sur un écran graphique une représentation exacte des pages qui seront imprimées. Ce genre d'outil permet de gagner un peu de temps et d'économiser du papier. L'utilisateur peut détecter les erreurs à l'écran mais il ne peut pas les corriger sur l'épreuve qui lui est présentée. Il doit se replonger dans la description du document et retrouver la commande ou la partie de texte à changer, comme il le fait lorsqu'il détecte une erreur sur une épreuve imprimée. Cela limite l'intérêt de l'outil.

Janus [Chamberlin] va plus loin dans ce sens. Il a été conçu dès le début comme un système intégrant un formateur de haut niveau et des moyens interactifs. Il utilise deux écrans (d'où son nom), l'un, alphanumérique, affichant la description du document dans le langage du formateur, l'autre, graphique, présentant l'image formatée. Les deux écrans affichent simultanément la même partie du document sous ces deux formes. Ainsi l'utilisateur retrouve rapidement la partie de la decription du document qui engendre une partie donnée de l'image formatée. Les modifications se font exclusivement sur la description du document et une commande particulière permet de les répercuter sur l'image formatée, au moment choisi par l'utilisateur. Il ne s'agit donc pas d'un vrai système interactif puisque d'une part l'effet des commandes est différé et que d'autre part l'image graphique ne peut pas être utilisée directement pour les modifications. Notons cependant que l'utilisateur peut apporter quelques changements au formatage effectué par le système, en agissant sur l'image formatée, mais c'est la seule possibilité de réelle interaction.

3.1.3. Les éditeurs syntaxiques

Les éditeurs syntaxiques, eux, sont en général de vrais systèmes interactifs. Bien qu'ils soient principalement destinés à la manipulation de programmes, les principes qu'ils mettent en œuvre sont d'un grand intérêt pour d'autres types de documents.

A la différence des éditeurs d'usage général mentionnés plus haut, les éditeurs syntaxiques connaissent le langage dans lequel est écrit le programme qu'ils traitent. Certains de ces éditeurs ne peuvent traiter qu'un langage. C'est le cas, par exemple, d'Adèle [Estublier] pour Pascal ou du Cornell Program Synthesizer [Teitelbaum], au moins dans sa première version destinée à un sous-ensemble de PL/1. D'autres, au contraire, peuvent être paramétrés. Ils acceptent une description formelle de la syntaxe et de la sémantique des langages de programmation et traitent alors des programmes écrits dans les

langages spécifiés de cette façon. Dans cette catégorie on trouve Mentor [Donzeau], Pecan [Reiss] ou Aloe [Medina] dans l'environnement Gandalf [Habermann].

Les éditeurs syntaxiques utilisent leur connaissance de la syntaxe du langage pour guider l'utilisateur dans la construction d'un programme correct. Ils peuvent bâtir un squelette de programme que l'utilisateur complète ensuite. Au cours de ce travail, ils proposent les instructions valides dans chaque environnement et vérifient que l'ensemble reste conforme à la grammaire du langage. De cette façon, ils réalisent la synthèse d'un programme à partir des éléments fournis progressivement par l'utilisateur. Mais ils sont également souvent capables d'analyse, acceptant des fragments de programme sous leur forme textuelle dont ils extraient la structure.

En fait les éditeurs syntaxiques ne constituent qu'un des outils proposés dans l'ensemble d'un environnement de programmation ou d'un atelier logiciel. Ils sont donc souvent accompagnés d'autres outils dont certains sont destinés à la production des documents qui accompagnent normalement les programmes, comme les spécifications ou les manuels. Ces outils peuvent être obtenus directement à partir de l'éditeur syntaxique lui-même, comme c'est le cas dans Mentor [Mélèse]. Ils peuvent aussi s'appuyer seulement sur l'environnement de l'atelier et constituer un outil plus indépendant, comme celui proposé par Concerto [Paris].

3.1.4. Les éditeurs et formateurs spécialisés

La plupart des systèmes présentés jusqu'ici sont d'abord conçus pour manipuler du texte. Or on trouve souvent dans les documents des éléments non-textuels : schémas, dessins, photographies, tableaux, formules mathématiques ou chimiques... Ce type d'élément pose deux problèmes distincts, l'un concernant leur intégration dans le corps du document, l'autre leur élaboration proprement dite.

L'intégration

Selon le type de système, l'intégration peut se faire de deux façons différentes.

L'élément à intégrer peut être créé et manipulé par le système lui-même. On trouve dans cette catégorie aussi bien des formateurs que des systèmes de traitement de texte. Dans le cas d'un formateur, le langage de description de document est complété pour permettre la description de formules (comme dans TEX [Knuth 84] ou Eqn [Kernighan 75]), de tableaux (comme dans TEX, Tbl [Lesk] ou Tblp [Dewey]), de graphiques (Grap [Bentley]) ou de schémas (Pic [Kernighan 82], Ideal [van Wyk]). Dans le cas d'un système interactif, le système comporte le plus souvent un (ou plusieurs) éditeurs spécialisés dans le traitement d'un type d'élément particulier. Les systèmes de traitement de texte sont

généralement limités dans ce domaine, et il faut plutôt chercher dans les sytèmes de traitement de documents techniques pour trouver ce type d'outil.

L'élément à intégrer peut être créé et manipulé par un outil spécialisé, extérieur au système de traitement de documents. On trouve, là aussi, des formateurs et des systèmes interactifs. Pour les formateurs, il s'agit le plus souvent d'insérer des images ou des graphiques qui proviennent d'autres applications (systèmes de CAO, scanners...). Ils permettent parfois de réduire ou d'agrandir l'élément de façon à lui donner des dimensions précises, en accord avec le format du document. L'élément à intégrer doit être représenté selon un mode standard, comme PostScript [Adobe] ou une norme graphique [Bono]. Pour les systèmes interactifs MacWrite [Williams] est un bon exemple. Des outils spécialisés permettent de traiter des dessins, des schémas, des formules et MacWrite peut accueillir ces éléments dans le texte et les redimensionner. Mais il n'autorise pas d'en modifier le contenu, et interdit de les insérer dans le fil du texte, au milieu d'une ligne. Si des modifications sont nécessaires, il faut reprendre l'outil adapté et réinsérer l'élément modifié.

La production

Les moyens de production de ces éléments varient selon la nature de l'élément à produire. Les formateurs n'acceptent pas la description de dessins ou de photographies, ils peuvent seulement intégrer de tels éléments, alors que les outils interactifs offrent les moyens de créer et modifier des dessins ou de retoucher des photographies numérisées.

Dans le domaine du graphique, on peut utiliser aussi bien certains formateurs que des systèmes interactifs [Foley]. Parmi les formateurs les plus répandus, c'est probablement Troff qui offre le plus de possibilités dans ce domaine, avec ses préprocesseurs Pic ou Grap, mais on peut aussi citer Tioga [Beach]. Cependant, les systèmes interactifs conçus pour illustrer les documents sont généralement d'un usage plus simple. Comme pour d'autres outils de production de documents, l'Alto [Thacker] a ouvert la voie avec Draw [Baudelaire], qui permet de décrire un schéma comme un ensemble d'éléments graphiques de base (rectangles, segments, cercles, ellipses, courbes...) positionnés les uns par rapport aux autres et dotés d'attributs (style et épaisseur de trait, texture de remplissage...). Cette approche a été reprise dans de nombreux autres outils qui ont mis l'accent soit sur le mode de manipulation du dessin (MacPaint et MacDraw notamment [Williams]), soit sur les relations qui lient les éléments [Joloboff 83].

Pour la présentation des tableaux, les formateurs offrent des possibilités intéressantes, alors que les systèmes interactifs proposent peu d'outils spécifiquement adaptés. Bien sûr, les outils graphiques peuvent être utilisés dans la construction de tableaux, mais, n'étant pas destinés à cet usage, ils rendent les manipulations un peu lourdes.

Dans le domaine des mathématiques on peut faire les mêmes constatations. Alors que TEX, et dans une moindre mesure Eqn, mais aussi Scribe et Mint [Hibbard], peuvent produire des formules mathématiques d'une bonne qualité typographique, l'utilisation des systèmes interactifs se révèle délicate. Ce point sera développé dans le chapitre suivant.

Lorsque les formateurs traitent des éléments non-textuels, ils le font toujours d'une façon très structurée, avec des commandes de haut niveau. On ne décrit pas un tableau en indiquant la place précise et la longueur de chaque filet ou de chaque chaîne de caractères, mais plutôt en décrivant son organisation en lignes et en colonnes, et en indiquant le contenu de chaque cellule. De même les formules ne sont pas décrites par des commandes de décalages, positionnement ou changement de corps, mais à travers leur structure mathématique.

3.1.5. Les interfaces utilisateur

Nous avons vu les principales fonctions offertes par les systèmes de production de documents. En plus de ces fonctions, un des aspects les plus importants de ces systèmes est l'interface qu'ils présentent à leur utilisateur. Dans le cas des formateurs, il s'agit du langage dans lequel l'auteur décrit l'organisation et/ou la mise en page de son document, ainsi que le texte et les différents éléments non-textuels qu'il contient. Pour les systèmes interactifs, il s'agit des commandes qui modifient le document et de l'image présentée à l'écran, à travers laquelle l'utilisateur voit le document.

De plus en plus, dans les systèmes interactifs, l'interface utilisateur est séparée des fonctions de traitement. Cela permet notamment de rendre les applications portables, en isolant dans les modules d'interface utilisateur les parties dépendant du matériel (affichage, saisie, désignation). Cela permet aussi d'adapter plus facilement l'application aux besoins de différentes catégories d'utilisateurs. Dans un premier temps, ces fonctions d'interface ont été développées spécifiquement pour des applications particulières. Maintenant on propose des outils d'usage général pour la construction d'interfaces utilisateur. Parmi les outils bien adaptés au traitement des documents on peut citer [Coutaz], [Gosling 84], [Kasik], [Morris 86], [Wong].

3.2. Scribe

Le but du projet Scribe, développé par B. Reid à l'université Carnegie-Mellon, est clairement énoncé dans [Reid 83]. Il s'agissait de fournir aux informaticiens un outil simple et relativement spécialisé, pour la production de documents attrayants et lisibles. Reid cherchait à tirer parti des imprimantes rapides à points tout en demandant un travail minimum à ses utilisateurs. Il voulait surtout leur éviter d'avoir à apprendre la typographie

ou un nouveau langage de programmation. Un autre objectif du projet consistait à fournir un mécanisme de communication de documents abstraits entre laboratoires de recherche.

Les principes de Scribe suivent les méthodes de préparation et de production des documents dans la chaîne éditoriale traditionnelle, qui met en jeu plusieurs métiers, chacun jouant un rôle particulier : l'auteur (ou les auteurs), le comité de lecture, l'éditeur, le directeur d'une collection ou le rédacteur en chef d'un journal, le maquettiste et le typographe. Scribe joue le rôle de plusieurs de ces professionnels. Il prend en charge la forme du document et laisse à l'utilisateur la seule responsabilité du contenu.

L'utilisateur travaille au niveau logique. Il décrit l'organisation du document, le document abstrait, et il exprime ses intentions, mais il ne spécifie pas les détails de la mise en page ou de la typographie. Toutes les informations liées à la présentation des documents sont contenues dans la base de données sur laquelle Scribe s'appuie pour produire l'image imprimée. Ainsi, en changeant d'imprimante, plusieurs résultats différents peuvent être obtenus à partir du même document : une imprimante à rosace n'offre pas les mêmes possibilités typographiques qu'une imprimante à laser. D'ailleurs B. Reid présente son système [Reid 80] comme un compilateur acceptant un langage de haut niveau, le langage dans lequel sont décrits les documents abstraits. Scribe peut produire différents documents concrets sur différentes imprimantes, de la même façon qu'un programme en Pascal peut donner lieu à différents codes objets pour différentes machines.

Le langage de description des documents est un langage déclaratif, donc non-procédural. L'utilisateur *décrit* en effet les éléments qui constituent son document, mais il n'indique pas la façon de les *traiter*, de les imprimer. C'est aussi un langage abstrait, puisqu'il ne se réfère à aucune machine ou imprimante particulière, mais seulement à la structure logique du document.

Ce langage, dont la syntaxe est simple, sert essentiellement à introduire des "marques" dans le texte d'un document. Ces marques peuvent délimiter des régions ou indiquer des commandes adressées au compilateur, mais la syntaxe est unique pour ces différentes fonctions. Les régions définies par des marques de début et de fin sont typées, et déterminent des "environnements". Un environnement peut porter sur un simple caractère, sur un mot, une expression, une phrase ou sur des éléments plus importants comme un paragraphe ou une page. A titre d'exemple, on peut citer les environnements italique, citation, exemple, énumération, description.

Les environnements utilisables sont définis dans la base de données. Celle-ci contient également la description des différents types de documents disponibles, qui représentent chacun l'environnement global qui peut être utilisé pour un document. La base de données Scribe propose de nombreux types de documents prédéfinis, tels que article, brochure, guide, lettre, manuel, rapport, thèse, etc... D'autres types de documents peuvent être

ajoutés, tout comme de nouveaux environnements, mais il s'agit là d'un travail particulier qui n'est pas du ressort des utilisateurs ordinaires du système, et qui requiert une bonne connaissance de la typographie.

Dans la base de données, chaque type de document et chaque environnement est décrit en termes d'attributs, sous la forme d'une suite de paires attribut-valeur (voir figure 1.1). Scribe connaît une centaine d'attributs qui définissent la police à utiliser, le corps des caractères, les marges, la justification, le mode de construction des lignes, les espacements verticaux, l'interligne, etc... Lors du traitement d'un document, le formatage est déterminé par les valeurs de ces attributs. Ces valeurs sont d'abord définies par le type du document, puis modifiées au cours du traitement en fonction des environnements rencontrés. Chaque environnement ne définit que quelques valeurs d'attribut, les valeurs des autres attributs sont héritées des environnements englobants ou du type du document. En effet, les environnements peuvent être imbriqués les uns dans les autres : on peut par exemple avoir un environnement italique dans un environnement énumération.

---

a) Définition de l'environnement Itemize dans la base de données :

```
@Define(Itemize, Break, Continue, Fill, LeftMargin +5, Indent -5,
        RightMargin -5, BlankLines break,
        Numbered <@@y[B]@, @@y[b]>, NumberLocation lfr,
        Spacing 1, Above 0.5lines, Below 0.5lines, Spread 0.5lines,
        Spaces compact)
```

b) Utilisation de l'environnement dans un document :

```
Cette figure contient trois parties :
@Begin(Itemize)
la définition de l'environnement dans la base de données,

l'utilisation de l'environnement dans un document,

le document formaté.
@End(Itemize)
```

c) Le document formaté.

Cette figure contient trois parties :

- la définition de l'environnement dans la base de données,
- l'utilisation de l'environnement dans un document,
- le document formaté.

Figure 1.1 : Un exemple, l'environnement Itemize de Scribe.

Le langage fournit également des commandes qui permettent de modifier certaines valeurs d'attribut soit pour l'ensemble d'un document (les commandes de modification sont alors en tête du document), soit pour une petite partie. Cela permet d'introduire des variations locales par rapport aux définitions de la base de données.

Scribe ne fournit pas seulement un moyen simple et puissant pour obtenir une typographie homogène et de bonne qualité, il effectue aussi certains traitements sur le document. En particulier, il gère les références croisées, les numérotations (chapitres, sections, pages, notes, etc...), il constitue les tables des matières et les index. Il est particulièrement bien adapté au traitement des documents scientifiques, et permet à chaque utilisateur de gérer sa propre base bibliographique. Il fournit de nombreux formats pour la présentation des références bibliographiques, conformes aux standards de la plupart des publications scientifiques.

Comme les compilateurs des langages de programmation, Scribe offre la possibilité de compilations séparées. Ainsi un gros document, ou un document rédigé par plusieurs auteurs, peut être formaté par parties, par exemple chaque chapitre indépendamment. Même dans ces conditions Scribe prend en charge l'ensemble du document et assure notamment que les références croisées et les numérotations sont cohérentes entre les différentes parties.

Malgré ces qualités, Scribe présente quelques difficultés pour le traitement des documents scientifiques. En particulier, il n'est capable, au moins dans sa version d'origine, de traiter que du texte. Les tableaux, les schémas, les formules mathématiques ne sont pas pris en compte. Cependant, la version commerciale de Scribe a remédié à certains de ces défauts et elle contient notamment une extension pour le traitement des formules mathématiques.

Les concepts mis en jeu dans Scribe ont inspiré le développement de nombreux formateurs, mais aussi de quelques systèmes interactifs, comme on le verra plus loin. Parmi les formateurs inspirés de Scribe on peut citer FOAM [Ganzinger], Scribble, [Woodman], Mint, LaTeX. Il y en a bien d'autres, mais deux d'entre eux présentent un intérêt certain : Mint et LaTeX.

3.2.1. Mint

Mint a été développé par P. Hibbard à l'université Carnegie-Mellon [Hibbard]. L'objectif initial de Mint était de préparer les outils pour des systèmes interactifs de composition de documents. Le formateur Mint reprend l'aspect externe de Scribe, c'est à dire essentiellement son langage, de sorte qu'un utilisateur de Scribe peut se servir de Mint immédiatement. Mais la structure interne du programme est très différente. Elle est

ouverte et permet d'étendre les possibilités de traitement du formateur, par exemple pour insérer des images dans les documents. La figure 1.2 donne un exemple de document en Mint.

```
&liredef(papier)
&grostitre(Les syst}mes pour la manipulation de documents
structur{s)
&centre(&i"Vincent Quint"
INRIA et LGI)

&soustitre(PRESENTATION)
De nombreux syst}mes sont aujourd'hui propos{s pour {crire
des documents en utilisant les ressources de l'ordinateur.
Ces syst}mes sont de types...

 ... partag{ en quatre cat{gories :
&debut(listenum)
les {diteurs et syst}mes de traitement de texte,

les formateurs,

les {diteurs syntaxiques,

les outils sp{cialis{s.
&fin(listenum)
Bien sur, ...
```

Figure 1.2 : Un document décrit dans le langage Mint.

L'architecture de Mint est fondée sur quatre types d'objets : les lignes, les boîtes, les colonnes et les pages. Les lignes contiennent une suite d'unités de composition élémentaires, des mots, mais aussi des expressions mathématiques ou des symboles. Les lignes sont assemblées pour former des boîtes représentant une unité textuelle continue, telle qu'un paragraphe, un titre ou la légende d'une figure. Les boîtes sont réparties dans des galées, qui sont de longs rouleaux de la largeur d'une page, contenant des éléments de même type. Ainsi les notes de bas de page, les schémas et le texte principal forment trois galées. Enfin, les lignes et les boîtes des galées sont réunies pour former des pages.

Les notions de galée et de boîte sont les principaux facteurs d'extensibilité et d'adaptabilité de Mint. Les boîtes permettent notamment de définir des mises en page sophistiquées. Mint peut également traiter des formules mathématiques.

3.2.2. LaTeX

TeX est reconnu comme étant l'un des meilleurs formateurs du point de vue de la qualité typographique des documents qu'il produit. Mais il se voit également souvent reprocher la complexité de son langage de description des documents. Ce langage est d'une grande richesse, ce qui permet de régler très finement les détails de présentation des documents ; mais cette richesse le rend en même temps difficile à maitriser par certains utilisateurs, notamment ceux qui n'ont pas de connaissances en typographie ni en programmation. Ce défaut est compensé par un mécanisme de macro-instructions qui permet de définir de nouvelles commandes d'un niveau plus élevé.

C'est cette possibilité qui a été mise à profit par [Lamport], en développant LaTeX. LaTeX peut être vu comme un formateur de haut niveau, très proche de Scribe, mais implanté sur TeX. Il bénéficie ainsi de la puissance et de la simplicité du langage de Scribe et de la qualité typographique de TeX. Comme TeX, il permet de décrire des formules mathématiques et des tableaux. Un environnement Picture autorise en plus la description de graphiques composés de lignes droites, de flèches, de cercles et de texte. Comme Scribe, il traite les bibliographies, les numérotations, les tables des matières, les index, les glossaires, etc...

Si LaTeX se distingue fortement de TeX dans la description des parties purement textuelles, les tables et les formules sont traitées pratiquement comme dans TeX, où ces objets sont déjà décrits d'une façon relativement structurée.

3.3. Interleaf

Différentes versions du système WPS (Workstation Publishing Software) [Morris 85] sont proposées par Interleaf pour les postes de travail les plus répandus (Sun, Apollo et VAXstation notamment). De nombreux aspects de ce système le rapprochent du Star [Harslem] dont il a visiblement subi l'influence.

WPS regroupe plusieurs fonctions : une gestion de fichiers à travers une métaphore de bureau et un traitement interactif de documents WYSIWYG, qui inclut des éditeurs de texte, de graphique et d'images.

3.3.1. Le gestionnaire de documents

Conformément à son modèle, le Star, WPS présente le classement des documents qu'il manipule selon l'image d'un bureau traditionnel : les différents documents sont rangés dans des dossiers, les dossiers sont regroupés dans des tiroirs et les tiroirs dans des armoires. Chacun de ces objets porte un nom choisi par l'utilisateur, qui est libre de créer, selon ses besoins, de nouveaux documents, dossiers, tiroirs et armoires. Un objet est représenté par une icône accompagnée de son nom. Toutes ces icônes sont réparties sur l'écran et l'utilisateur peut mettre un peu d'ordre dans son "bureau" en les déplaçant à l'aide de la souris. Cette métaphore, qui a également été reprise sur le Macintosh, est maintenant assez répandue.

Tous les objets sont manipulés avec les mêmes commandes que le contenu des documents. Il est possible de déplacer des documents d'un dossier à un autre, ou de déplacer des dossiers d'un tiroir à un autre, par exemple. On peut également copier et détruire des documents ou des objets de plus haut niveau, de la même façon qu'on déplace, copie ou détruit des parties de documents.

Lorsque l'icône d'un document est visible sur l'écran, on peut ouvrir le document, comme on ouvre une armoire ou un tiroir. Un fenêtre est créée sur l'écran et la première page du document y est affichée. On peut alors lire et modifier le contenu du document, on est dans l'éditeur.

3.3.2. Le modèle de document

L'éditeur considère le document comme un ensemble ordonné de *composants* typés. Les *types* peuvent être choisis et créés par l'utilisateur. Ainsi, pour une lettre, on utilisera les types date, adresse de destination, salutation, paragraphe, et signature. Pour un rapport, on utilisera les types titre, auteur, résumé, titre de section, titre de sous-section et paragraphe.

L'utilisateur a toute liberté de créer selon ses besoins de nouveaux types et d'insérer dans le document, en tout endroit, un composant de n'importe quel type. Les types ne sont pas là pour structurer réellement les documents, mais plutôt pour assurer une certaine homogénéité de leur présentation. En effet, à chaque type est associé un ensemble de *propriétés*, qui sont en fait des attributs de présentation, spécifiant comment les composants de ce type doivent être affichés et imprimés. Mais il est possible de modifier les propriétés d'un composant particulier, de façon qu'il se présente différemment des autres composants du même type.

3.3.3. Le formatage des documents

L'essentiel du travail de formatage est effectué automatiquement à partir des propriétés. Celles-ci ne sont pas seulement attachées aux types de composants, mais également à l'ensemble du document et aux pages. Les propriétés attachées au document sont très limitées. Elles indiquent seulement si la coupure des mots en bout de ligne doit se faire et sur quelle imprimante doit sortir le document.

Au contraire, les propriétés des pages sont plus riches. Elles décrivent le format et les dimensions des pages : orientation, hauteur, largeur, marges droite et gauche, espaces de haut et de bas de page, position et style des numéros de page, texte et position des titres courants de haut et de bas de page. Ces propriétés des pages peuvent varier au long du document.

Chaque composant a également un ensemble de propriétés qui définissent, outre son nom, son formatage. Ces propriétés sont divisées en trois groupes. Le premier groupe indique comment le texte du composant doit être découpé en lignes :

- les marges : espace au-dessus de la première ligne du composant, en dessous de sa dernière ligne, à sa droite et à sa gauche.
- le retrait de la première ligne,
- l'interligne : distance verticale entre deux lignes successives,
- le mode d'alignement des lignes : à gauche, à droite, centré, justifié,
- les caractéristiques typographiques des caractères : style, corps, graisse et italique,
- la coupure des mots : coupure demandée ou non.

Le deuxième groupe de propriétés concerne les tabulations, leur position (distance de la marge gauche) et leur type. Si une ligne comporte des caractères de tabulation, les chaînes de caractères délimitées par ces marques se placent par rapport aux positions de tabulation successives, selon le type des tabulations :

- gauche : le début de la chaîne de caractères se place sur la tabulation,
- droite : la fin de la chaîne de caractères se place sur la tabulation,
- centré : le centre de la chaîne de caractères se place sur la tabulation,
- numérique : le point décimal des nombres, ou leur dernier chiffre s'il n'y a pas de point, se place sur la tabulation.

Le dernier groupe de propriétés des composants concerne leur position dans la page :

- nouvelle page : le composant doit commencer ou non en haut d'une nouvelle page,
- contrôle de veuve : le nombre minimum v de lignes du début du composant qui doivent figurer dans la même page ; s'il n'y a pas assez d'espace pour v lignes en bas de la page, le composant commence en haut de la page suivante.
- contrôle d'orphelin : nombre minimum de lignes à la fin du composant qui peuvent être séparées du reste du composant par un saut de page.
- coupure : indique si un saut de page peut avoir lieu dans le composant,
- saut de page : indique si un saut de page doit avoir lieu après le composant.

Toutes ces propriétés associées à chaque élément sont automatiquement appliquées par l'éditeur au cours de la manipulation du document et l'image affichée correspond en permanence à ces propriétés. On remarquera donc qu'on a sur l'écran à tout moment l'image fidèle de ce qui sera obtenu sur l'imprimante, y compris le découpage du document en pages.

3.3.4. L'édition du document

Pour créer un document, on peut partir de zéro en créant de nouveaux types, avec toutes leurs propriétés, au fur et à mesure des besoins. On crée ensuite des composants conformes à ces types. Mais la méthode la plus adaptée consiste sans doute à créer des documents particuliers, qui sont des squelettes de documents typiques, comportant au moins un composant de chacun des types possibles. On peut avoir ainsi un squelette de lettre, un squelette de rapport, etc... La création d'un document commence alors par la copie du document squelette correspondant au document à créer. Il reste à remplir les composants vides en tapant leur texte, et à créer de nouveaux composants avec les types prédéfinis. La structure du document définie par le squelette n'est qu'indicative : l'utilisateur peut parfaitement créer de nouveaux types, modifier les propriétés d'un type existant, supprimer des composants qui étaient dans le squelette, ou encore changer leur ordre ou leur présentation.

Il y a plusieurs façons de se déplacer dans un document. Une méthode évidente est de faire défiler le document sur l'écran, dans un sens ou dans l'autre : une barre sur le côté droit du document permet, avec la souris, de contrôler ce défilement. Il est aussi possible de faire des recherches de chaînes de caractères en avant ou en arrière. On peut enfin utiliser les noms de type des composants et ainsi sauter au prochain (ou précédent) composant d'un type donné.

Plusieurs opérations de manipulation du document supposent qu'un composant (ou plusieurs) soit sélectionné. Cette sélection se fait grâce à une zone qui s'étend sur tout le côté gauche du document, et où apparaissent les noms de type des composants, en face de leur texte dans le document. Dans cette zone, on peut sélectionner des composants en les désignant avec la souris. Mais il est aussi possible, par des menus, de sélectionner tous les éléments d'un type donné, par exemple. Une fois que des composants sont sélectionnés, différentes opérations peuvent leur être appliquées :

- Couper : la partie sélectionnée est retirée du document et rangée dans le presse-papiers. Elle pourra ensuite être "collée" ailleurs.
- Copier : la partie sélectionnée est simplement copiée dans le presse-papiers sans être supprimée du document.
- Changer : le composant sélectionné change de type et les propriétés attachées au nouveau type lui sont appliquées.
- Propriétés : certaines (ou toutes) les propriétés du composant peuvent être modifiées et les modifications peuvent s'appliquer à ce seul composant ou à tous les composants de même type.
- Retours Chariot : dans la partie sélectionnée, le découpage des lignes est figé, ou, s'il était déjà figé, il est libéré.

Si, au lieu de sélectionner un ou plusieurs composants, on pointe avec la souris dans la zone affichant les types à gauche du document, entre deux composants, on peut alors créer de nouveaux composants. Un menu listant tous les noms de composants existant dans le document permet de choisir le type du nouveau composant. Le composant créé prend alors les propriétés du premier composant du document portant ce nom. Des composants successifs peuvent être fusionnés pour former un seul composant, même s'ils sont de types différents ; dans ce cas c'est le type du premier composant qui est conservé, mais les textes provenant des autres composants gardent leurs attibuts typographiques (style, corps, graisse...), qui peuvent encore être changés.

Le texte des composants est manipulé d'une façon désormais classique dans ce type de système. Pour insérer, on place le point d'insertion avec la souris et tout le texte frappé au clavier vient s'insérer à cet endroit. Au fur et à mesure de la frappe, le texte reste présenté suivant les propriétés du composant où il s'insère. Pendant cette insertion, et quelles que soient les propriétés du composant, on peut modifier les attributs typographiques des caractères entrés (style, corps, graisse, italique). Pour les modifications, la sélection du texte s'opère avec la souris et la partie sélectionnée s'affiche sur fond noir. On peut "couper" ou "copier" la partie sélectionnée, on peut aussi changer ses attributs typographiques.

Les commandes "couper" et "copier" permettent, on l'a vu, de ranger des extraits du document dans le presse-papiers. Le contenu de ce tampon peut ensuite être réinséré dans le texte par une simple commande "coller".

3.3.5. L'éditeur graphique

L'éditeur de documents d'Interleaf inclut un éditeur graphique qui permet de manipuler des *diagrammes*. Un diagramme peut être soit un dessin libre soit la représentation graphique de données numériques. Le dessin libre est de type structuré : un diagramme est constitué d'une collection d'objets de base qui ont des relations entre eux. Cette approche est comparable à celle de Draw [Baudelaire], ou plus encore à celle de [Joloboff 83].

Les objets de base sont les segments de droite, les rectangles, les ellipses, les polygones, les courbes et les textes. Ces objets ont des *propriétés* qui définissent l'épaisseur de leur trait ou la texture de remplissage. Ils peuvent être groupés pour former des ensembles qui sont alors manipulés d'un bloc. Chaque objet ou groupe d'objets peut subir de nombreuses opérations comme le changement de ses propriétés, de sa taille, de sa position ou de son orientation. Il peut également être copié.

Pour faciliter les positionnements, l'utilisateur peut faire apparaître (et disparaître) une *grille* de points régulière qui lui sert à aligner les différents objets lors de leur création avec la souris. Au moment de la désignation d'un point d'un objet en cours de création, le système peut, sur option, retenir le point de la grille le plus proche de la position courante de la souris, ce qui assure un alignement parfait des objets. Une autre facilité offerte pour le positionnement des éléments est la *gravité*. Il est possible d'affecter une gravité plus ou moins importante à un objet, de sorte qu'il "attire" les objets de son voisinage, selon leur distance.

L'intégration de l'éditeur de diagrammes et de l'éditeur de texte reste limitée. Les chaînes de caractères qui apparaissent dans un diagramme ne peuvent pas subir tous les traitements du texte du document et inversement, toutes les facilités, de positionnement notamment, offertes par l'éditeur de diagrammes ne peuvent pas être utilisées dans le texte principal du document. L'intégration des diagrammes dans le document se fait à travers des *cadres* rectangulaires que l'on peut placer n'importe où dans le texte du document, y compris dans une ligne de texte. Lorsqu'on édite dans un cadre, on est dans l'environnement des diagrammes, lorsqu'on édite le texte principal du document, les cadres sont traités comme des boîtes noires qui, lorsque les changements du texte le nécessitent, sont simplement déplacées sans jamais être coupées en bout de ligne ou en bas de page.

A l'éditeur graphique s'ajoute un éditeur d'images, pour le traitement des photographies numérisées. Cet éditeur permet de retoucher les images avec toute une série de pinceaux différents ou pixel par pixel, grâce à un zoom. Il permet aussi de faire varier le contraste des images ou de les inverser (positif-négatif) en jouant graphiquement sur la fonction de transfert. Enfin, les images peuvent être agrandies, réduites, tournées ou déplacées.

3.3.6. Conclusion

WPS est typiquement un système dans la lignée des outils d'édition développés initialement sur l'Alto [Thacker] et diffusés ensuite avec le Star [Harslem]. Il se caractérise par une interface utilisateur particulièrement soignée, où les menus et les feuilles de propriétés tiennent une grande place bien que les commandes les plus courantes puissent également être entrées au clavier. Cette interface facilite grandement l'apprentissage et l'utilisation du système.

Le système d'Interleaf a suscité à son tour le développement d'outils analogues, tels que MacAuteur [Icon] ou Pléiade [Nanard 86]. D'autres, comme Frame Publisher, ont étendu ses possibilités dans le domaine de la mise en page, ajoutant des fonctions que l'on trouve, par exemple, dans Page Maker.

Le modèle de document est simple, donc facile à appréhender par l'utilisateur, qui, grâce aux notions de composant et de propriété, se trouve déchargé d'une large partie de la tâche de mise en page. Les propriétés autorisent une certaine uniformité de la présentation, sans toutefois garantir une homogénéité absolue. L'utilisateur ne se sent pas contraint par un modèle limité ou trop rigide, et peut, à tout moment, tout modifier : le contenu du document, mais aussi les types et les propriétés.

Bien que les propriétés évoquent les environnements de Scribe, WPS n'offre pas la richesse de la structure de Scribe : il n'y a aucune numérotation automatique, aucune gestion des références, des index ou des tables des matières. L'article présentant le système [Morris 85], composé bien sûr avec WPS, est d'ailleurs révélateur : il contient deux sections portant le même numéro (VII) !

3.4. Les normes régissant les documents

La multiplication des systèmes de production de documents ainsi que l'évolution des imprimantes et autres appareils d'affichage a rendu très difficile l'échange de documents sous leur forme électronique. Chaque système a son propre format de représentation, chaque imprimante a son propre jeu de commandes. Pour résoudre ce problème, les

organismes de normalisation ont défini des modes de représentation des documents, qui, s'ils sont mis en œuvre dans les produits à venir, devraient faciliter les échanges entre systèmes différents.

Il existe deux catégories de normes de représentation des documents. Les unes concernent les documents formatés ; elles décrivent des pages destinées seulement à être imprimées ou affichées sur un écran. Les autres s'intéressent à des documents révisables, dont il est possible de modifier le contenu et l'organisation [Joloboff 86]. Il faut en plus mentionner les normes touchant au domaine du graphique [Bono], qui peuvent décrire certaines parties des documents, mais qui ne sont pas présentées ici.

3.4.1. Les normes pour les documents révisables.

L'ISO a défini deux normes concurrentes pour la représentation des documents révisables : SGML [ISO 85] et ODA [ISO 86]. A ces deux normes internationales reconnues, il faut ajouter une proposition, Interscript [Ayers], qui, bien que n'ayant pas été retenue par les organismes de normalisation, présente de nombreux points intéressants. Interscript n'est pas présenté dans cette partie sur les normes.

SGML (Standard Generalized Markup Language) est dans la lignée des langages de marquage logiques comme GML [Goldfarb]. Dans son esprit, il est également proche de Scribe. Il considère un document comme une structure abstraite formée d'éléments de types divers : articles, chapitres, paragraphes, figures, légendes, etc... Différentes applications peuvent traiter cette structure, des formateurs, mais aussi des systèmes documentaires, par exemple. Le marquage (l'information ajoutée au texte même du document) délimite et identifie les éléments de la structure et leur associe des attributs. Il est essentiellement déclaratif. Les traitements sont définis par les applications qui se basent sur la description fournie par SGML ; ils ne sont pas indiqués dans le document. C'est ce qui permet à plusieurs applications de travailler sur la même description du document. Il est toutefois possible d'insérer dans le document certaines procédures de traitement destinées à un système particulier, mais elles sont alors clairement isolées du reste du document.

Le marquage utilisé est défini formellement pour chaque type de document. Comme dans une grammaire formelle, des définitions de types indiquent quels éléments et quels attributs peuvent figurer dans un document, et dans quel ordre ils peuvent apparaître. Ces définitions de types permettent de spécifier des classes de documents et de vérifier si un document donné est bien conforme au modèle de sa classe. Elles permettent aussi un marquage incomplet des documents, puisque les marques absentes peuvent être retrouvées de façon non ambiguë à partir des définitions.

La norme définit un méta-langage pour l'introduction de nouveaux symboles : des types d'éléments et des attributs. Avec la syntaxe concrète proposée par SGML, ces déclarations de types et les descriptions de documents peuvent être produites "à la main" et sont lisibles pour l'œil humain. SGML n'impose pas l'utilisation de logiciels spécifiques et peut même être considéré comme un moyen d'échange de documents entre être humains, par exemple un auteur et un éditeur. Le très court exemple suivant le montre.

```
<!ELEMENT
1 these        - -  (titre, auteur, chapitre+)
2 titre        O O  (#CHARS)
3 auteur       - O  (#CHARS)
4 chapitre     - O  (titrechap, paragraphe+)
5 titrechap    O O  (#CHARS)
6 paragraphe   - O  (#CHARS)
>
```

D'après cette définition exprimée dans le méta-langage de SGML, une thèse est formée d'un titre, d'un auteur et d'une suite de chapitres. Le titre et l'auteur sont chacun constitués d'une simple chaîne de caractères. Chaque chapitre contient un titre et une suite de paragraphes. Avec cette définition, et en utilisant la syntaxe concrète de SGML, une thèse est représentée de la façon suivante :

```
<these>
Une approche de l'édition structurée des documents
<auteur> V. Quint
<chapitre>
Typographie et informatique
<paragraphe>
La première partie de ce chapitre est consacrée...
<paragraphe>
La typographie et l'informatique sont ...
 ...
<chapitre>
Une première approche
<paragraphe>
...
</these>
```

Sur cet exemple très simplifié, on peut remarquer que certaines marques n'apparaissent pas dans le document. Ce sont les marques implicites indiquées dans la définition de type. En effet, les symboles - et O dénotent les marques qui peuvent être omises (O) et celles qui doivent nécessairement figurer dans le document (-). La première colonne concerne la

marque de début de l'élement, la deuxième colonne concerne la marque de fin. La marque de fin se distingue de la marque de début par le caractère '/' après '<'.

Un chapitre commence toujours par un titre de chapitre ; il n'y a donc pas de marque explicite de début de titre de chapitre. Pour la même raison, il n'y a pas non plus de marque de début pour le titre de la thèse. La plupart des marques de fin sont omises, puisque la marque de début de l'élément suivant indique sans ambiguïté la fin du précédent. Seule la thèse elle-même nécessite une marque de fin explicite.

A la différence de SGML, ODA (Office Document Architecture) prend en compte le formatage des documents, en plus de leur structure logique et de leur contenu. Plus précisément, ODA permet de décrire

- des documents formatés qui ne peuvent qu'être affichés ou imprimés selon les intentions exprimés par l'émetteur,
- des documents révisables non formatés, qui sont destinés à être édités et formatés par le récepteur,
- des documents révisables et formatés qui peuvent être soit imprimés tels qu'ils sont, soit édités et reformatés par le récepteur.

Dans le dernier cas, le contenu du document est unique et partagé par la structure logique et la structure de mise en page (nous dirons par la suite "la structure physique"). Ces deux structures présentent une certaine homogénéité, au moins dans l'exposé de la norme. Toutes deux sont des structures arborescentes et les objets, aussi bien logiques que physiques, qui composent un document ont des relations hiérarchiques.

Les classes dans ODA jouent le même rôle que les types dans SGML. Elles représentent un ensemble d'objets qui ont les mêmes caractéristiques. Les objets qui composent un document particulier sont des instanciations de ces classes. Le document lui-même est un objet et il existe donc des classes de documents. Tout comme les types de SGML, les classes de ODA ne sont pas définies par la norme, qui offre seulement les moyens de les spécifier. Cette spécification se fait à travers des structures génériques logiques et physiques qui décrivent les classes d'objets logiques et physiques. Un document particulier a une structure spécifique logique et/ou une structure spécifique physique qui sont conformes à ces structures génériques. Les structures génériques assurent la cohérence des structures spécifiques et servent à guider la création et le formatage des documents.

Bien que le vocabulaire soit différent, il n'y a pas de différence fondamentale entre la partie logique de ODA et SGML. Les différences se situent évidemment au niveau physique, qui n'est pas pris en compte par SGML. Alors que les objets des structures logiques d'ODA présentent une certaine uniformité (ils sont divisés en objets de base,

décrivant le contenu, et en objets composés, qui sont tous les autres objets), les objets des structures physiques sont très variés ; ils se répartissent en cinq catégories :

- les ensembles de pages, qui représentent par exemple toutes les pages d'un chapitre,
- les pages de base, dont le contenu n'est pas structuré,
- les pages composées, qui sont formées de cadres et de blocs,
- les cadres, qui délimitent des zones dans une page composée ou dans d'autres cadres (les cadres peuvent former une structure sur plusieurs niveaux),
- les blocs, qui, dans les pages composées, déterminent la position où doit s'afficher le contenu du document.

Cette structure relativement complexe sert à décrire l'organisation physique des pages qui constituent le document. Elle est employée aussi bien au niveau générique qu'au niveau spécifique. Les attributs qui peuvent porter sur ces objets définissent notamment leurs dimension et position, mais d'une façon rigide puisqu'il s'agit exclusivement de constantes.

Qu'il s'agisse de la structure logique ou physique, les feuilles des arbres sont des éléments de contenu, qui peuvent être de nature textuelle, géométrique ou photographique. D'autres natures (ODA dit "architectures") de contenu seront ajoutées par la suite. Chaque architecture de contenu a sa propre structure, qui est organisée différemment de celle du document.

Alors que SGML permet de décrire les structures logiques des formules mathématiques ou des tableaux, ODA ne considère pas ce genre d'éléments. Il semble que l'approche retenue soit de les représenter à l'avenir avec des structures de contenu spéciales, bien que les outils existants au niveau logique soient utilisables. Il est vrai qu'au niveau physique, ODA ne permet pas une description correcte de ce genre d'éléments.

Malgré certaines différences entre ODA et SGML, il peut sembler étrange que les organismes de normalisation proposent simultanément deux normes pour la même fonction, la représentation des documents. A cause de leurs différences, les domaines d'application ne seront probablement pas les mêmes. SGML sera sans doute d'avantage utilisé dans le domaine de l'édition et de la photocomposition, alors que ODA s'orientera plutôt vers les applications de bureau. Mais ces deux domaines ne sont pas complètement disjoints, et il arrive qu'un document saisi d'abord avec des outils bureautiques soit ensuite destiné à une publication plus large, et doive être traité par des typographes. Il faudra donc des passerelles entre les deux normes...

3.4.2. Les normes pour les documents formatés.

ODA, on vient de le voir, peut représenter des documents formatés, en plus des documents révisables. D'autres normes existent pour la représentation de documents formatés. Elles peuvent être classées en deux catégories :

- les formats statiques,
- les formats dynamiques.

La structure physique définie par ODA représente un format statique. Les pages, les cadres et les blocs sont décrits par des positions et des dimensions constantes, exprimées dans une unité fixe (2,54/1200 mm !). Dans la structure physique spécifique, tout est figé. On peut également classer dans cette catégorie la norme T. 73 du C.C.I.T.T. [CCITT] qui s'appuie sur les mêmes concepts de cadres que ODA.

Avec les formats dynamiques, ce ne sont pas les pages elles-mêmes qui sont décrites, mais la façon de les produire. On utilise également le terme de "langage procédural de description de pages" [Reid 86], car il s'agit réellement de langages de programmation, qui permettent de décrire des algorithmes dont l'éxécution produit l'image d'une page. Trois langages de cette famille ont été définis, ce sont Interpress [Xerox], PostScript [Adobe] et DDL (Document Description Language). A vrai dire, ce ne sont pas des normes au même titre que celles que nous avons déjà présentées. Ces langages n'ont pas été définis ni retenus par les organismes de normalisation, mais ils constituent des normes de fait employées dans un nombre croissant de systèmes de traitement de documents. Tous trois sont fondés sur des principes analogues et sont assez voisins. PostScript étant actuellement le plus répandu, il sera seul présenté ici.

PostScript est principalement utilisé pour décrire des pages destinées à une imprimante, bien que certains systèmes l'emploient également pour afficher des images sur écran. Les imprimantes qui travaillent avec PostScript sont formées de deux parties distinctes. Une machine de marquage reçoit un signal video représentant l'image d'une page, et marque cette image sur le papier. Il s'agit le plus souvent d'une imprimante à laser, mais il existe aussi des photocomposeuses travaillant en PostScript. La deuxième partie est un ordinateur spécialisé qui reçoit des programmes PostScript, les interprète (PostScript est un langage interprété) et produit, comme résultat de l'éxécution du programme, des images de pages sous forme de matrices de points. Ce sont ces matrices de points qui, lorsque la page est complète, sont envoyées sous forme de signal video vers la machine de marquage.

L'exécution d'un programme PostScript a lieu sur une machine à pile. L'interprète range dans la pile les unités lexicales qu'il reçoit, jusqu'à ce qu'il rencontre un nom de procédure connu. Il exécute alors cette procédure en prenant sur la pile les paramètres

d'appel et il retourne dans la pile le résultat de la procédure. Le langage est postfixé (d'où son nom), puisque les paramètres sont envoyés avant l'appel de la fonction.

Le langage contient de nombreuses fonctions graphiques mais aussi des fonctions de calcul. Il permet de définir des procédures et donc de s'étendre lui-même. Les primitives graphiques sont fondées sur un modèle qui s'inspire du processus sérigraphique. L'image est produite en appliquant de l'encre sur une feuille de papier à travers un stencil [Warnock 82]. L'encre peut être de couleur unie ou elle peut elle-même être une image (tout se passe alors comme si on projetait une diapositive à travers le stencil). Le stencil est décrit soit sous la forme de trous dont le contour est défini par une suite de droites et de courbes (des cubiques), soit sous la forme d'une matrice de bit, chaque bit à 1 représentant un trou élémentaire dans le stencil. Indépendamment de la forme du stencil, la région de la feuille de papier où l'encre est déposée peut être réduite par un contour défini. Plusieurs stencils, plusieurs encres et plusieurs régions peuvent être utilisées successivement pour juxtaposer ou superposer des images sur la même feuille.

Ce modèle est utilisé aussi bien pour imprimer des photographies numérisées, des graphiques ou des caractères, qui sont des formes comme les autres. Il faut ajouter que de nombreuses transformations géométriques sont possibles sur toutes les formes, grace à une matrice qui combine des translations, des homothéties et des rotations. Tout cela fait de PostScript un langage extrêment puissant.

A la différence des langages de programmation courants, ce sont souvent des programmes (des systèmes de production de documents) qui produisent du PostScript, et pas seulement des programmeurs. Dans un document décrit en PostScript, il y a généralement en tête une partie fixe, qui a été écrite "à la main", et qui est systématiquement insérée par l'application de traitement de documents. Cette partie, spécifique de l'application, définit les unités utilisées, ajoute quelques procédures (par exemple de justification ou de tracé de cadres), spécifie des caractères spéciaux, etc... Après vient la partie variable qui décrit le document lui-même, en faisant appel aux fonctions standard de PostScript et aux fonctions définies dans l'en-tête.

PostScript a été implanté dans plusieurs appareils d'impression, depuis les imprimantes laser de bas de gamme jusqu'aux photocomposeuses. Il est également utilisé pour représenter les sorties d'un nombre croissant de systèmes de production de documents. C'est maintenant une norme de fait bien établie, qui offre une réelle indépendance vis à vis des appareils de sortie, tout en assurant la meilleure qualité sur chaque appareil et une grande fidélité d'un appareil à l'autre.

4. CONCLUSION

L'ensemble des problèmes qui concernent la production de documents est vaste et varié. La tendance qui se dégage, au moins dans le domaine de la bureautique, est d'intégrer dans un système unique plusieurs outils jusque là séparés. C'est notamment le cas des éditeurs-formateurs ou encore des systèmes qui, comme Frame Maker, ajoutent aux précédents des outils de mise en page élaborés. Dans le domaine de la typographie des fonctions en nombre croissant sont confiées à l'ordinateur, et on peut noter une évolution de la représentation des documents vers un plus haut degré d'abstraction.

Mais tous les problèmes ne sont pas résolus. En particulier, la jonction entre les outils bureautiques et les outils typographiques est difficile. Les fichiers produits par les traitements de texte, ou même par les éditeurs-formateurs plus récents, ne sont pas acceptés par les systèmes de photocomposition. Un langage comme PostScript permettrait de résoudre le problème, mais cette solution suppose que tout le travail typographique est fait en amont, dans le système qui produit du PostScript. Actuellement ces systèmes travaillent soit en mode "batch", et ils sont mal admis comme outils de bureau, soit en mode interactif, et leurs possibilités typographiques sont souvent limitées et mises entre les mains d'utilisateurs peu avertis.

Une autre solution est proposée par SGML, et dans une moindre mesure par ODA. Mais il n'existe pas (pas encore) d'outils confortables pour créer des documents dans ces formats. SGML propose d'utiliser un simple éditeur pour entrer directement la description du document. Cette méthode est peu acceptable dans le monde de la bureautique. ODA sera utilisé dans une première phase comme moyen d'échange entre systèmes de traitement de texte conventionels et donc avec une représentation logique très limitée, insuffisante pour le typographe.

L'idée d'une représentation abstraite des documents, que l'on trouve dans SGML et dans la partie logique de ODA, mais aussi dans Scribe et ses descendants, permet de résoudre la plupart de ces problèmes. Mais il reste à définir et à construire les outils qui travaillent sur ce type de représentation. Certains sont déjà disponibles pour le formatage (Scribe, Mint, LaTeX), mais pour la création des documents rien de réellement satisfaisant n'existe sur le marché.

En s'inspirant des systèmes que nous avons déjà vus, on peut définir l'outil idéal. Il s'agit d'un système interactif qui affiche sur l'écran une image du document de bonne qualité, sur laquelle l'utilisateur peut agir directement, comme avec les éditeurs-formateurs récents. Il travaille sur des documents structurés et tire parti de cette structure pour guider et assister l'utilisateur, comme le font les éditeurs syntaxiques. Il n'exige pas que l'utilisateur joue à la fois le rôle de l'écrivain, de l'éditeur, du maquettiste et du typographe. Il laisse l'auteur exprimer ses intentions et permet à chacun de jouer son rôle

au cours de la vie du document dans la chaîne éditoriale. Pour une utilisation "bureautique", il intègre autant que possible l'expertise des différents métiers des arts graphiques.

Un tel outil n'existe pas et sa conception présente de nombreuses difficultés. Aussi avons-nous commencé par l'étude d'un sous-problème, celui de l'édition des formules mathématiques. C'est seulement après cette expérience que nous avons conçu un système général, Grif, qui traite les documents dans leur globalité et qui se fonde sur un modèle de document adapté. Le prochain chapitre présente l'étude sur l'édition des formules mathématiques et les deux suivants décrivent le modèle de document de Grif puis le système qui le met en œuvre.

CHAPITRE 2

UNE PREMIERE APPROCHE : LE TRAITEMENT DES FORMULES

1. PRESENTATION

A la fin des années 70 on pouvait constater avec des systèmes comme Scribe [Reid 80] tout l'intérêt de l'approche structurelle pour le traitement des documents complexes. D'autre part, les systèmes interactifs, et notamment les machines de traitement de texte, avaient montré la facilité d'utilisation due à l'interactivité. Mais sauf dans quelques domaines particuliers, comme le génie logiciel ou la conception assistée par ordinateur, les systèmes offrant à la fois l'interactivité et la structuration n'existaient pas. Il nous a paru intéressant de mener une expérience qui permettrait de réunir, dans un même système, ces deux approches.

Pour une première étude, nous avons retenu le traitement des formules mathématiques, un domaine où il semblait nécessaire d'apporter des solutions efficaces à des problèmes réels, et où l'expérience acquise à cette occasion pourrait servir de base à des développements plus ambitieux.

1.1. Les outils existants

Schématiquement, l'auteur qui veut écrire un document mathématique a deux types d'outils à sa disposition : les formateurs et les systèmes de traitement texte [Palais]. Certains formateurs, notamment TEX, ou Troff complété de son préprocesseur Eqn, sont effectivement capables de traiter des documents contenant de nombreuses formules mathématiques. Leur inconvénient est cependant bien connu : ils ne permettent pas à l'utilisateur de contrôler directement ce qu'il écrit. Celui-ci doit d'abord maitriser le langage de description des documents et des formules, et, au moment où il écrit, il ne dispose pas d'outils pour vérifier qu'il ne commet pas d'erreurs syntaxiques et qu'il obtiendra bien ce qu'il veut dans la forme finale du document. De nombreuses épreuves sont généralement nécessaires pour obtenir un résultat satisfaisant et les contraintes syntaxiques ne facilitent pas l'expression de la pensée.

Les quelques exemples suivants montrent la syntaxe imposée par les formateurs ; ils donnent la représentation, dans les langages de différents formateurs, de la formule[1] suivante :

$$\frac{1}{2n}\int_0^{\sqrt{y}}\left(\sum_{k=1}^{n}\sin^2 x_k(t)\right) f(t)\,dt \qquad (1)$$

Scribe :

```
@Over(Num 1,Denom 2n)@~@Int(From 0, To @Sqrt(y))
@~(@Sum(From k=1, To n)@~@Sin@up[2]x@down[k](t))f#(t)dt
```

Mint :

```
@fract(1,2n) @int(0,@sqrt(y),@paren(@sum(k=1,n,@sup(sin,2)
@sub(x,k)(t)))f(t)dt)
```

TEX :

```
{1\over 2\pi}\int_0^{\sqrt y}\bigglp\sum_{k=1} ^n
\sin^2x_k(t)\biggrp f(t)\,dt
```

Un palliatif a été proposé pour raccourcir le cycle édition-épreuve-relecture, avec les systèmes qui permettent d'obtenir rapidement, sur écran, une épreuve du document formaté. TEX, par exemple, dispose de plusieurs outils de ce type, comme ceux proposés par [Aiello] ou [Agostini], ou encore TEX Preview. Avec ce type d'outil, l'utilisateur peut voir, en général simultanément sur deux écrans ou deux fenêtres d'un même écran, le document formaté et le programme qu'il a écrit pour produire ce document. L'image du document formaté est construite par le formateur quand l'utilisateur le désire, mais elle ne peut pas être recomposée à chaque modification élémentaire, les temps de calcul du formateur étant trop importants.

Un autre type d'aide à l'utilisation des formateurs est représenté par les outils comme Stratec [Foata], qui utilisent des claviers étendus et ont une certaine connaissance du langage de formatage qu'ils éditent. L'extension du clavier permet essentiellement de

---

[1] Cette formule est une forme simplifiée de l'exemple bien connu des utilisateurs de TEX :

$$\frac{1}{2\pi}\int_{-\infty}^{\sqrt{y}}\left(\sum_{k=1}^{n}\sin^2 x_k(t)\right)(f(t)+g(t))\,dt$$

Elle a été simplifiée pour alléger l'exposé. En TEX, sous sa forme complète, elle s'écrit :

```
$$ {1\over 2\pi}\int_ {-\infty} ^ {\sqrt y}\bigglp\sum _ {k=1} ^n
\sin^2x_k(t)\biggrp\biglp f(t)+g(t)\bigrp\,dt $$
```

frapper en une seule touche les mots-clés représentant les lettres grecques ou les symboles mathématiques courants. La connaissance du langage se limite à une partie de la syntaxe, ce qui autorise un contrôle automatique, à la saisie, du parenthésage et de la structure de bloc (portée des changements de polices notamment). Ce type d'outil permet, à un coût raisonable, d'éviter les difficultés les plus courantes dans la frappe des textes destinés à un formateur, mais ne change pas fondamentalement la nature des formateurs.

Une alternative aux formateurs est présentée par certains systèmes interactifs de traitement de texte qui offrent quelques facilités complémentaires pour l'inclusion de formules dans les documents qu'ils traitent. Mais si la contrainte syntaxique des formateurs n'existe pas avec ces systèmes, une autre lui a été substituée, la contrainte géométrique. En effet, les formules sont généralement traitées comme du graphique sans structure, ou très peu structuré. La machine offre un choix plus ou moins large de caractères dits scientifiques et permet de les disposer en tout point de l'écran, simplement en déplaçant le curseur et en entrant chaque caractère à la position courante du curseur. On compose ainsi une sorte de mosaïque qui peut effectivement représenter n'importe quelle formule, pour peu que le choix de caractères soit suffisant. Le grand avantage par rapport au formateur est que l'utilisateur voit en permanence ce qu'il fait et qu'il peut réagir immédiatement en cas d'erreur de sa part. Il n'a pas de surprise lors de l'impression de son document, qui est généralement une copie conforme de ce qu'il a vu sur l'écran.

Malheureusement plusieurs défauts sont inhérents à ce principe. La première difficulté est que l'auteur doit se faire une représentation bidimensionnelle précise de la formule qu'il veut entrer : il doit dimensionner lui-même les parties de taille variable des formules, comme les barres de fraction ou les grandes parenthèses. Il doit aussi positionner précisément chaque élément. Mais une autre difficulté surgit lorsqu'il souhaite modifier une formule existante. Même un changement mineur peut avoir des conséquences nombreuses sur l'aspect graphique d'une formule. Supposons par exemple que l'on veuille effectuer la transformation suivante :

$$z = \frac{1}{\sqrt{x + y}} \longrightarrow z = \frac{1}{\sqrt{x^2 + y}}$$

Dans l'esprit de l'auteur il s'agit tout simplement d'insérer l'exposant 2 après le caractère x. Mais en termes graphiques, cela revient à une longue série de modifications élémentaires, à la charge de l'utilisateur :

décaler vers le bas l'expression sous le radical pour laisser la hauteur de l'exposant,
prolonger le radical vers le bas,
décaler vers la droite la partie qui suit x pour laisser la largeur de l'exposant,
prolonger vers la droite la barre horizontale du radical,

prolonger également le trait de fraction vers la droite, puisque le dénominateur s'allonge,
décaler le '1' du numérateur vers la droite pour qu'il reste centré,
placer enfin le petit '2'.

Pour obtenir le même résultat, les modifications peuvent sembler plus simple sur la forme linguistique utilisée par les formateurs, plus proches de la logique de l'utilisateur. En effet, avec Eqn, la formule précédente, avant modification, s'écrit :

```
z = 1 over sqrt x+y
```

et après modification :

```
z = 1 over sqrt { x sup 2 +y }
```

La modification consiste donc simplement à ajouter `sup 2` après `x`, sans oublier toutefois d'insérer des accolades avant `x` et après `y` et de placer judicieusement les espaces (avant `sup` et après `2`). Mais la difficulté avec un formateur n'est pas tant de savoir *ce* qu'il faut changer que de savoir *où* il faut opérer le changement. Il faut d'abord retrouver, dans la longue description du document, l'endroit où se trouve la partie à changer, et, précisément pour les formules, ce n'est pas chose aisée puisqu'un développement mathématique est habituellement constitué d'une suite de formules relativement semblables où il n'est par facile de repérer celle qui doit être changée. Les systèmes affichant sur l'écran une épreuve du document ne sont pas d'un grand secours, puisqu'ils ne permettent pas de faire le lien entre un point de l'image formatée (là où une erreur a été décelée par l'utilisateur) et l'instruction à modifier dans la description du document. Cependant un formateur comme TEX fournit les mécanismes de base qui permettraient d'établir cette relation entre le texte source et l'image.

Un autre inconvénient majeur de l'approche "mosaïque" tient à la représentation de la formule qui est très pauvre et ne permet aucun traitement autre que l'affichage sous une forme identique et sur un appareil ayant les mêmes caractéristiques que l'écran d'édition. Le formateur, lui, offre plus de souplesse, puisqu'il représente la formule sous une forme plus logique, à partir de laquelle il est notamment possible de construire plusieurs images différentes pour des appareils aux capacités graphiques variées.

1.2. Une autre approche

Puisqu'aucun des outils disponibles n'apporte vraiment la solution, mais que chacun a des avantages, nous avons essayé d'en définir un nouveau, qui cumule les avantages des précédents, sans toutefois présenter aucun de leurs inconvénients.

Pour construire un tel système, nous avons retenu des formateurs l'approche structurée qui est proche de l'utilisateur, mais nous avons cherché à supprimer la syntaxe rigide du langage de description de formules. Des systèmes de traitement de texte nous avons conservé le haut degré d'interactivité, et surtout la possibilité d'agir *directement* sur l'image de la formule affichée sur l'écran. Nous avons de plus essayé de débarasser l'utilisateur de la tâche fastidieuse de dessiner les formules élément par élément.

Ces principes ont guidé la conception et la réalisation d'Edimath, un éditeur interactif de formules mathématiques structurées. En plus de l'intérêt pratique de construire un système bien adapté à l'édition des mathématiques, cette expérience permettait d'aborder le domaine de l'édition structurée des documents sans attaquer d'emblée un problème très vaste (celui de l'édition de documents complets), mais tout en cherchant, dans le domaine limité des formules, des solutions de portée plus générale.

Les formules constituent en effet un échantillon représentatif des problèmes qui se posent dans le traitement des documents. Du point de vue typographique, on y trouve les principales difficultées :

- de multiples jeux de caractères, certains alphabétiques, mais pas uniquement latins, d'autres non-alphabétiques, pour les nombreux symboles mathématiques, certains utilisateurs créant même leurs propres symboles,
- plusieurs corps (au moins trois) utilisés simultanément dans chaque formule, avec des gros caractères pour des symboles comme le $\sum$ de la sommation, et des petits caractères pour les indices et exposants, entre autres,
- toutes sortes de styles de caractères, chaque style portant une indication sémantique : italique pour les variables et paramètres, romain pour les noms de fonction, mais aussi des styles moins usuels comme le gothique ou celui employé pour représenter certains ensembles courants ($\mathbb{R}$ $\mathbb{N}$ $\mathbb{C}$ $\mathbb{Z}$).
- des éléments que l'on peut assimiler à du graphique : barres de fraction, radicaux, grandes parenthèses, et en général tous les symboles dont la taille varie avec celle des expressions mathématiques qu'ils délimitent,
- des positionnements très variés, comprenant l'alignement habituel du texte, mais aussi des centrages (numérateur/dénominateur) des décalages (indice ou exposant), des alignements verticaux (colonne de matrice).

Du point de vue du système informatique, les formules concentrent également les principales difficultés d'un traitement interactif et structuré. La structure est riche et variée et l'interaction doit y jouer un rôle important.

2. LA STRUCTURE DES FORMULES

La principale caractéristique d'Edimath est de traiter des formules structurées. Nous présentons ici les structures prises en compte par le système.

2.1. Les principes de structuration

Il y a plusieurs façons de considérer la structure d'une formule. Les machines de traitement de texte décrites plus haut considèrent la structure géométrique des formules, de façon à reproduire leur image. A l'opposé, les compilateurs considèrent la structure mathématique dans ses moindres détails, de façon à calculer les formules. Dans Edimath, la structure des formules est élaborée en fonction de deux facteurs principalement :

- les traitements effectués par un système de manipulation de documents,
- la simplicité d'utilisation d'un système fondé sur cette structure.

Si l'on reste dans le domaine des manipulations de documents, un éditeur doit essentiellement permettre de saisir une représentation des formules sur laquelle les modifications soient aisées et à partir de laquelle la restitution des formules soit possible sur différents appareils. Il n'est pas nécessaire que le système ait une connaissance approfondie de la signification mathématique de la formule qu'il manipule, puisqu'il ne fait pas de traitement mathématique comme peuvent en faire les compilateurs ou les systèmes de calcul formel. Il lui faut cependant le minimum de connaissance sur la structure mathématique pour produire un affichage correct, puisqu'on veut décharger l'utilisateur de ce travail de présentation. En effet, l'aspect graphique d'une formule est précisément fait pour que la structure mathématique apparaisse clairement au lecteur. C'est même là la caractéristique fondamentale de la notation mathématique. On peut donc concevoir un système qui, à partir de la structure mathématique, construit l'image d'une formule.

Il reste à donner au système une connaissance "typographique" des mathématiques qui lui permette d'afficher une formule d'après sa structure. Pour cela, la structure doit comporter toute l'information nécessaire à la construction de l'image de la formule.

Comme cette structure représente la base sur laquelle reposent les traitements de l'éditeur, elle doit être fournie par l'utilisateur. Plus la structure est complexe et riche, plus les traitements sont élaborés, mais aussi plus la manipulation est délicate pour l'utilisateur. Il suffit de constater la lourdeur de l'interface des systèmes de calcul formel. Il faut donc trouver un compromis entre la richesse de la structure et la simplicité de sa manipulation pour l'utilisateur.

Nous avons retenu une structure minimum qui ne prend en compte que les éléments structurels qui affectent le positionnement des différentes expressions qui constituent une formule. Toute structure mathématique qui s'écrit sous la forme de caractères alignés les uns à la suite des autres est ignorée. Ainsi, pour Edimath, l'équation $y = ax + b$ n'a pas d'autre structure que celle d'une chaîne de caractères, même si on peut lui reconnaître une structure mathématique faisant intervenir une égalité, une somme, un produit, des variables et des paramètres.

Dans une expression aussi simple que $y = ax + b$, on doit, en bonne typographie, tenir compte de la structure, par exemple pour augmenter les espaces de part et d'autre du signe = ou de l'opérateur +. On doit aussi mettre en italique les variables et les paramètres. Tout cela nécessite une connaissance de la structure plus fine que celle de la simple chaîne de caractères. Mais si on demande à l'utilisateur de décrire lui-même la structure de la formule, ce niveau de détail imposerait un dialogue très lourd entre l'utilisateur et le système. C'est pourquoi, toute structure qui est habituellement représentée sous la forme d'une suite de caractères est obtenue, quand elle est nécessaire pour l'affichage, par analyse de la chaîne de caractères entrée au clavier. En d'autres termes, l'utilisateur n'a à frapper que les six touches correspondant aux six caractères composant la formule $y = ax + b$ pour obtenir une typographie correcte dans la forme imprimée.

Cela est d'une certaine façon comparable au fonctionnement des éditeurs syntaxiques destinés à la manipulation des programmes : l'utilisateur travaille généralement au niveau structurel pour l'organisation de l'ensemble du programme et des instructions complexes, mais, pour les détails les plus fins, comme les expressions arithmétiques notamment, il travaille au niveau des chaînes de caractères, l'éditeur effectuant l'analyse de ces chaînes pour en extraire la structure syntaxique.

Dès que la notation mathématique habituelle demande un positionnement différent du simple alignement de caractères, il faut de toute façon que l'utilisateur le signale lorsqu'il entre une formule. Plutôt que de lui demander d'indiquer un positionnement géométrique, on lui demande d'indiquer la construction mathématique qui provoque ce positionnement. Cela n'est pas plus complexe pour lui ; au contraire, il peut ainsi s'exprimer à un niveau logique qui est sans doute plus proche de sa pensée. Et pour le système c'est un moyen simple et efficace de prendre en compte la structure de la formule.

2.2. Les constructions d'Edimath

Ces principes étant posés, nous définissons maintenant la structure précise des formules traitées par Edimath. Chaque formule est structurée sous une forme arborescente. La racine de l'arbre représente la totalité de la formule et les nœuds du premier niveau représentent les *constructions* mathématiques successives qui composent la formule de gauche à droite, en considérant qu'on change de construction à chaque fois qu'on rencontre autre chose qu'un simple alignement de caractères. Ainsi, la formule suivante :

$$\frac{1}{2n}\int_0^{\sqrt{y}}\left(\sum_{k=1}^{n}\sin^2 x_k(t)\right) f(t)\,dt$$

est constituée au premier niveau de quatre constructions :

une fraction,
une intégrale (le signe d'intégration et ses deux limites),
une expression entre deux grandes parenthèses,
une chaîne de caractères : f(t) dt.

Les niveaux inférieurs de l'arbre représentent le détail de la structure de chacune de ces constructions (à l'exception de la chaîne de caractères qui n'est pas décomposée) : le numérateur et le dénominateur pour la fraction, les deux limites pour l'intégrale, le contenu de la parenthèse. Tous ces éléments contiennent des *expressions* mathématiques, c'est à dire des suites de constructions, exactement comme la formule elle-même ; ils sont donc structurés de la même façon que la formule. Cette décomposition peut se poursuivre récursivement jusqu'aux feuilles de l'arbre, qui sont des chaînes de caractères.

Avec l'exemple de la formule (1) on a déjà vu certaines constructions. Pour Edimath, en plus de la chaîne de caractères, nous avons retenu en tout huit constructions, qui doivent permettre de décomposer n'importe quelle formule ou expression mathématique. En accord avec les principes énoncés précédemment, il ne s'agit pas toujours de constructions mathématiquement complètes, comme on a déjà pu le voir avec l'intégrale. Chaque construction a un ou plusieurs *constituants*, spécifiques du type de la construction, et chacun de ces constituants peut contenir soit une expression quelconque (une suite de constructions) soit un simple symbole, selon le type du constituant. La liste complète des constructions d'Edimath avec leurs constituants est donnée brièvement ci-dessous, avant d'être détaillée :

exposant : un seul constituant,
indice : un seul constituant,
fraction : deux constituants, le numérateur et le dénominateur,
racine : un seul constituant,

intégrale : deux constituants, les deux bornes,
triple : trois constituants, principal, inférieur et supérieur,
vecteur : un nombre variable de constituants, les éléments,
bloc : trois constituants, les deux symboles délimiteurs et le contenu.

L'exposant est une construction à un seul constituant, formé d'une expression quelconque. Cette construction permet de représenter une expression qui est mise en position d'exposant après une autre construction. Elle est le plus souvent utilisée pour représenter une puissance, mais d'autres utilisations sont possibles, comme dans l'expression C_n^p, qui représente un nombre de combinaisons et où le caractère *p*, en position d'exposant, n'est pas une puissance. La construction exposant est relative à la construction qui la précède (le caractère C dans l'exemple du nombre de combinaisons) dans la structure de la formule. Elle ne peut donc pas apparaître comme première construction d'une expression.

L'indice est comparable à l'exposant. Il n'a qu'un constituant qui peut être une expression quelconque et qui se place en position d'indice après la construction précédente. Mais si la construction précédente est un exposant, l'indice s'applique à la construction qui précède l'exposant, ce qui permet d'affecter à la même construction à la fois un indice et un exposant. Pour augmenter la souplesse du système, dans le cas où une construction porte à la fois un indice et un exposant, on accepte un ordre quelconque entre les deux constructions indice et exposant. Donc, si la construction qui précède un exposant est un indice, l'exposant se rapporte à la construction qui précède l'indice.

Plutôt que de définir, comme on vient de le voir, deux constructions à un seul constituant, l'exposant et l'indice, qui se rapportent à une construction précédente, on aurait pu penser à une seule construction qui rassemble trois constituants : l'exposant, l'indice et l'expression sur laquelle ils portent, l'exposant ou l'indice étant facultatifs. Cette construction serait plus proche de la structure mathématique. Dans une des premières versions d'Edimath cette construction à trois constituants a effectivement été utilisée, mais elle a finalement été remplacée par les deux constructions indice et exposant parce qu'elle imposait à l'utilisateur de délimiter le début de la construction sur laquelle portaient l'indice et/ou l'exposant, ce qui s'est révélé malaisé à l'usage. La solution qui a été finalement retenue est peut-être plus complexe à gèrer pour l'éditeur, mais elle est plus confortable pour l'utilisateur.

La **fraction** est une construction à deux constituants, le numérateur et le dénominateur, qui peuvent être des expressions quelconques. Elle correspond exactement à la construction mathématique fraction.

La racine est une construction à un seul constituant, qui peut être une expression quelconque. Elle correspond exactement à la racine carrée des mathématiques. Une construction racine avec un constituant supplémentaire, qui serait l'ordre de la racine, permettrait de représenter également des racines cubiques ou toute autre racine. Mais dans l'état actuel d'Edimath, la racine n'admet qu'un constituant, et c'est sans doute une des limitations actuelles du système. Il faut cependant noter qu'il existe un palliatif qui consiste à introduire un exposant avant la racine pour obtenir $\sqrt[3]{\ }$.

L'intégrale est une construction à deux constituants, les limites inférieure et supérieure de l'intervalle d'intégration, qui peuvent être toutes deux des expressions quelconques. Les noms donnés aux constituants, comme dans d'autres constructions, ne correspondent pas complètement à l'utilisation qui peut en être faite. Ainsi, pour une intégration sur un domaine (par exemple $\int_\Delta$), le nom du domaine (Δ) occupe la place habituellement occupée par la limite inférieure de l'intervalle d'intégration. Pour l'intégrale, comme pour l'indice et l'exposant, la structure retenue n'est pas mathématiquement complète puisqu'elle n'inclut pas la fonction intégrée ni la variable d'intégration. Là encore, c'est le confort de l'utilisateur qui a primé, puisque la prise en compte de la fonction intégrée aurait nécessité que l'utilisateur délimite explicitement cette fonction.

On n'a pas non plus retenu comme constituant le type de l'intégrale : intégrale simple, double, triple, curviligne, etc... La construction intégrale représente une intégrale simple. Mais, comme les deux limites peuvent être omises, une intégrale double peut être représentée par deux intégrales dont la première n'a pas de limites. De même, une intégrale triple peut se représenter avec trois intégrales. En revanche il n'y a pas de moyen de représenter une intégrale curviligne, puisque le symbole intégration lui-même ne peut pas être spécifié : c'est toujours $\int$. Notons cependant que dans la version Macintosh ce problème est résolu par l'ajout d'un composant optionnel, qui est un simple caractère se superposant au symbole $\int$.

Le triple, comme son nom l'indique, est une construction à trois constituants. Cette construction est utilisée pour représenter des constructions mathématiquement différentes mais graphiquement semblables. Elle sert à représenter toutes les constructions qui s'écrivent avec une expression ou un symbole sur la ligne principale (appelé constituant principal du triple), une expression ou un symbole au-dessous (appelé constituant inférieur) et/ou une expression ou un symbole au-dessus (constituant supérieur). Ainsi, chacune des formules ci-dessous est représentée par une construction triple :

$$\sum_{k=1}^{n} \qquad \lim_{x \to \infty} \qquad \overline{A + B} \qquad \xrightarrow{f(x)} \qquad \overrightarrow{AB} \tag{2}$$

La construction triple peut dans certains cas sembler redondante avec d'autres constructions, notamment la fraction et l'intégrale. L'intégrale, comme la construction triple dans de nombreux cas d'utilisation, représente un symbole avec un constituant en haut et un autre en bas. Mais on notera que les constituants inférieur et supérieur de triple sont centrés au-dessus et au-dessous du constituant principal, alors que les limites de l'intégrale sont placées à la droite du symbole $\int$. D'autre part la hauteur du constituant principal d'une construction triple est indépendante de son environnement, alors que la hauteur du symbole intégrale doit varier selon l'expression qui suit.

On pourrait aussi penser à utiliser la construction triple pour représenter une fraction, en prenant pour constituant principal de triple une simple barre. Mais il faut noter que les constituants inférieur et supérieur de triple sont écrits avec des petits caractères, ce qui n'est pas la convention la plus usuelle pour les numérateurs et dénominateurs de fraction.

Le vecteur est une construction comportant un nombre variable de constituants, appelés éléments du vecteur, qui peuvent être des expressions quelconques. Cette construction permet de traduire tout ce qui s'écrit sous la forme d'un alignement vertical d'expressions. On peut utiliser le vecteur pour représenter une colonne de matrice, ou encore plusieurs formules affichées l'une sous l'autre. Dans la formule[1] ci-dessous, cette construction est utilisée au premier niveau de la structure pour représenter l'empilement des trois lignes principales, mais elle est aussi utilisée, dans la parenthèse de la troisième ligne, pour représenter l'empilement des deux expressions sous le deuxième $\sum$.

$$
\begin{aligned}
G(z) = e^{\ln G(z)} &= \exp\left(\sum_{k>1} \frac{S_k z^k}{k} \right) = \prod_{k>1} e^{S_k z^k/k} \\
&= \left(1 + S_1 z + \frac{S_1^2 z^2}{2!} + \ldots\right)\left(1 + \frac{S_2 z^2}{2} + \frac{S_2^2 z^4}{2^2.2!} + \ldots\right)\ldots \qquad (3) \\
&= \sum_{m>0} \left(\sum_{\substack{k_1,k_2,\ldots k_m>0 \\ k_1+2k_2+\ldots+mk_m=m}} \frac{S_1^{k_1}}{1^{k_1}.k_1!} \frac{S_2^{k_2}}{2^{k_2}.k_2!} \ldots \frac{S_m^{k_m}}{m^{k_m}.k_m!} \right) z^m
\end{aligned}
$$

La construction vecteur est utilisée pour représenter les matrices, comme celles des formules (4a) et (4b), même si certaines matrices peuvent poser des difficultés. En effet, Edimath ne dispose pas d'une construction spécifique et une matrice doit être décrite soit sous le forme d'un unique vecteur soit sous la forme d'une suite de vecteurs. Avec une suite de vecteurs, on n'obtient de bons résultats que dans le cas où dans chaque ligne de la

[1] Cette formule est également un exemple célèbre, tiré de la documentation de Eqn. On se reportera à la documentation UNIX pour en avoir la description dans le langage Eqn, trop longue pour être reproduite ici sans erreur...

matrice tous les éléments ont la même hauteur : un élément plus haut ou plus petit que les autres romprait l'alignement horizontal, puisque la notion de ligne n'existe pas. Mais comme les vecteurs introduisent la notion de colonne, des éléments de la même colonne peuvent avoir des longueurs différentes, comme c'est le cas de la formule (4a).

$$A = \begin{pmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22}+1 & a_{32} \\ a_{13} & a_{23} & a_{33} \end{pmatrix} \qquad (4a)$$

On utilise un unique vecteur pour les matrices dont la hauteur des différents éléments d'une même ligne est variable. Chaque ligne de la matrice constitue un seul élément du vecteur, ce qui garantit un alignement correct dans les lignes. Le problème se pose alors pour les alignements verticaux des colonnes de la matrice. Il faut aligner les colonnes en ajoutant des espaces entre les éléments d'une même ligne de la matrice. C'est de cette façon qu'a été obtenue la matrice de la formule (4b).

$$A = \begin{pmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & \frac{a_{22}}{2} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{pmatrix} \qquad (4b)$$

Le **bloc** est la dernière des huit constructions d'Edimath. C'est une construction à trois constituants, qui permet de représenter toute expression entre parenthèses, crochets, accolades, etc... L'un des constituants, le contenu du bloc, est une expression quelconque, les deux autres sont des symboles : le symbole ouvrant et le symbole fermant. Cette construction est utilisée dans les formules (1), (3) et (4). Elle est également utilisée dans la formule (5), où le symbole ouvrant est une accolade, le contenu un vecteur et le symbole fermant un blanc.

$$\text{signe}(x) = \begin{cases} 1 \text{ si } x>0 \\ 0 \text{ si } x=0 \\ -1 \text{ si } x<0 \end{cases} \qquad (5)$$

En plus de ces huit constructions, Edimath utilise des **chaînes de caractères** comme on l'a déjà vu. A chaque chaîne de caractères est associé l'alphabet auquel appartiennent les caractères. Il y a donc une nouvelle chaîne de caractères à chaque changement d'alphabet. Ainsi la simple formule $c = 2\pi r$ est constituée de trois chaînes de caractères : la chaîne 'c = 2' dans l'alphabet standard, la chaîne 'p' dans l'alphabet grec et la chaîne 'r' à nouveau dans l'alphabet standard. L'alphabet standard est l'ISO IA5, à l'exclusion de toute variante. La difficile question de la normalisation des jeux de caractères n'a évidemment pas été résolue par Edimath, qui permet seulement de traiter plusieurs jeux de

caractères sans imposer de code particulier à chaque caractère. Cependant, il est clair que pour être compréhensibles d'une installation à une autre, les formules traitées par Edimath doivent être associées aux codes des jeux de caractères utilisés.

En plus des huit constructions de base et des chaînes de caractères, Edimath fait intervenir des *symboles* dans les structures des formules. Les symboles sont précisément des caractères spéciaux qui doivent être connus d'Edimath pour être traités correctement. La lettre 'π' de l'exemple de la circonférence du cercle n'est pas considérée comme un symbole, puisqu'elle ne requiert aucun traitement particulier ; elle est seulement affichée à la suite du caractère précédent et son code n'a pas besoin d'être connu d'Edimath : qu'elle soit représentée par le code ASCII de 'p' ou par celui de 'q' dans l'alphabet grec n'a d'importance qu'au niveau de l'affichage. Tant qu'on utilise les mêmes codes (c'est là que se situe la question de la normalisation) la formule est affichée de la même façon.

Au contraire des caractères, les symboles ont un code imposé par Edimath, qui les traite spécifiquement. Il y a deux catégories de symboles : les opérateurs de grande taille et les symboles élastiques. Les opérateurs de grande taille sont des caractères de plus grande dimension que les autres, mais dont la taille ne dépend pas du contexte. Edimath reconnaît comme symboles de grande tailles les quatre suivants, qui sont surtout utilisés comme constituant principal de constructions triples : Σ Π ∪ ∩. Ces symboles ont été distingués des caractères essentiellement pour limiter le nombre de jeux de caractères nécessaires. Leur grande taille aurait imposé un jeu de caractères supplémentaires, alors que dans la plupart des versions d'Edimath ils sont affichés en utilisant plusieurs caractères de taille normale, ce qui impose donc un traitement spécifique et justifie que leur code soit défini.

Les symboles élastiques sont des caractères dont la taille varie avec le contexte dans lequel ils se trouvent. Ce sont d'une part tous les symboles utilisés dans les constructions bloc : les parenthèses, accolades, crochets, barres verticales, etc... : ils prennent la hauteur de l'expression qu'ils englobent. Ce sont d'autre part les symboles utilisés dans les constructions triples et qui prennent la longueur de la construction triple où ils apparaissent : les barres horizontales ou les flèches qui surmontent une expression dont elles prennent la longueur, comme dans la formule (2). Tous ces symboles doivent bien être connus en tant que tels pour que l'éditeur sache qu'il doit calculer leur taille.

Ces changements de *taille* des symboles (élasticité) doivent être distingués des changements de *corps* des caractères, changements qui sont liés à la structure. Les constructions de base induisent des changements de corps automatiques, par exemple pour les indices et exposants qui s'affichent dans un corps plus faible que l'expression à laquelle ils se rapportent. Ce type de changement de corps s'applique à tous les caractères, quel que soit leur alphabet, et aussi aux symboles, qui, tout en restant relativement gros ou élastiques, seront plus ou moins fins, selon la construction ou le constituant où ils apparaissent.

Si on compare la structure prise en compte par Edimath avec celle utilisée par les formateurs qui traitent les formules mathématiques, on constate qu'elle en est très proche. En effet, ce sont les mêmes critères qui ont présidé à la définition des structures dans tous ces systèmes. La structure doit être suffisamment riche pour permettre la construction automatique d'une bonne image des formules ; elle doit aussi être suffisament simple pour que l'utilisateur n'ait pas trop de difficultés à la décrire.

3. LES DIFFERENTES REPRESENTATIONS DES FORMULES

La structure que l'on vient de présenter est la base sur laquelle est construit Edimath. Mais dans Edimath, comme dans tout éditeur structuré, différentes représentations des objets manipulés coexistent, chacune étant adaptée à un type de traitement particulier. L'ensemble est maintenu cohérent de façon que toutes les représentations décrivent effectivement les mêmes objets. Edimath travaille sur trois représentations :

- une représentation interne, pour les manipulations structurelles,
- une représentation graphique, pour l'affichage et le dialogue avec l'utilisateur,
- une représentation externe, pour le stockage en mémoire secondaire.

3.1. La représentation interne

Toutes les manipulations de structure s'effectuent sur un arbre qui constitue la représentation *interne* d'une formule en cours de traitement. Cet arbre correspond à la représentation abstraite de la formule et décrit les relations structurelles entre les constructions et constituants qui interviennent dans la formule. A titre d'exemple, la figure 2.1 représente l'arbre de la formule (1).

Les règles de construction de l'arbre de la représentation interne sont basées sur les règles de structuration. Il y a essentiellement deux règles de construction de l'arbre :

1- Les constructions qui constituent une expression sont représentées par une suite ordonnée de nœuds de même niveau attachés au nœud de la construction ou du constituant qui les contient. La figure 2.1 montre deux expressions formées de plusieurs constructions : la formule elle-même et le contenu de la grande parenthèse. L'ordre des nœuds des constructions dans l'arbre est celui dans lequel les constructions apparaissent, de gauche à droite, dans la forme graphique de la formule. Le seul cas où cet ordre est indéterminé est celui d'un indice et d'un exposant qui se rapportent à une même construction. Dans ce cas les nœuds de l'indice et de

$$\frac{1}{2n}\int_0^{\sqrt{y}}\left(\sum_{k=1}^{n}\sin^2 x_k(t)\right) f(t)\,dt$$

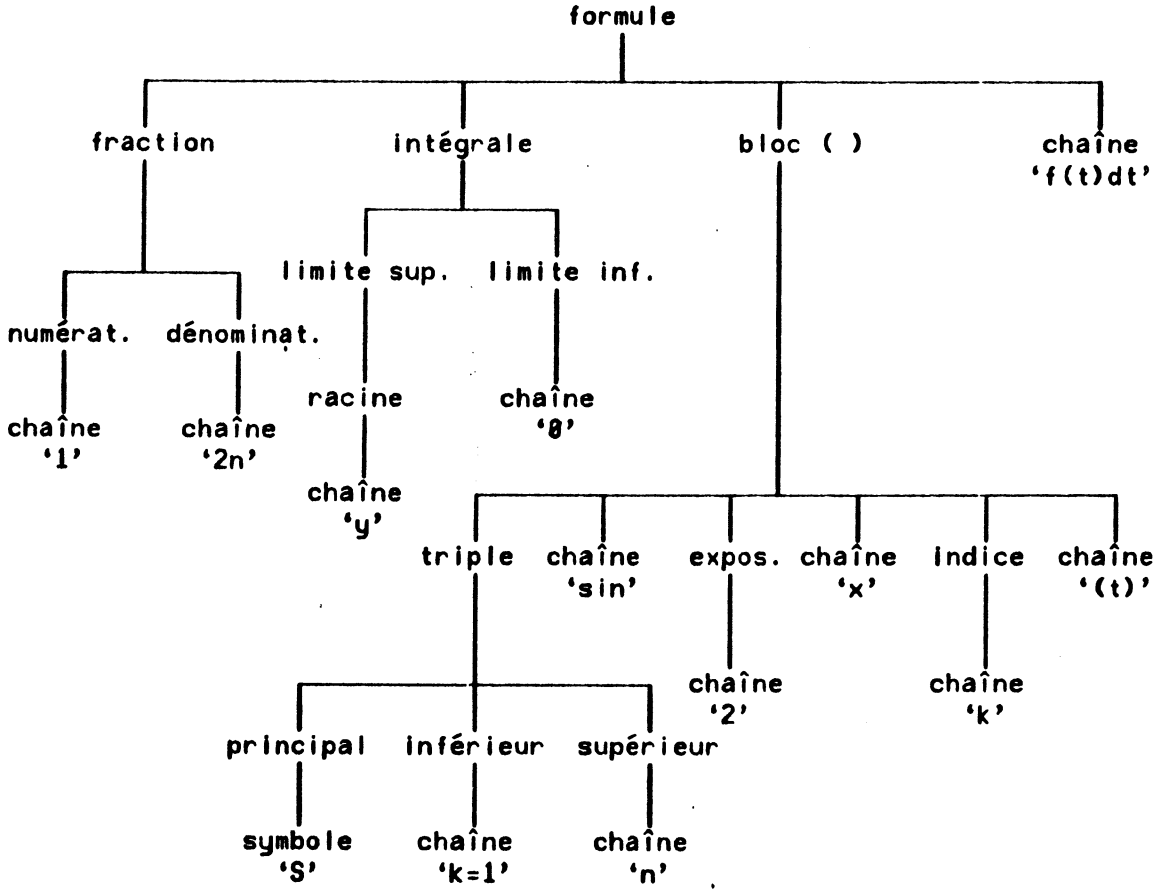


Figure 2.1 : une formule et sa représentation interne.

l'exposant peuvent être placés dans un ordre quelconque, immédiatement après le nœud de la construction à laquelle ils se rapportent (voir la figure 2.2 qui montre deux arbres équivalents pour la même formule).
Cette règle s'applique également à un constituant ou une construction qui ne contient qu'une seule construction. Le nœud de cette unique construction est attaché directement à celui du constituant ou de la construction qui la contient. C'est le cas dans la figure 2.1 notamment de la chaîne '1' dans le numérateur ou de la racine dans la limite supérieure de l'intégrale.

2- Les constituants d'une construction sont représentés par des nœuds de même niveau attachés au nœud de leur construction. C'est le cas dans l'exemple de la figure 2.1 du numérateur et du dénominateur de la fraction, des limites de l'intégrale ou encore des trois constituants du triple. L'ordre des nœuds des constituants d'une même

construction n'a théoriquement pas d'importance. Cependant, pour simplifier les traitements, ils sont toujours rangés dans le même ordre : numérateur avant dénominateur, limite inférieure d'intégrale avant la limite supérieure (contrairement à ce qui apparaît sur la figure 2.1 pour des raisons esthétiques), etc...

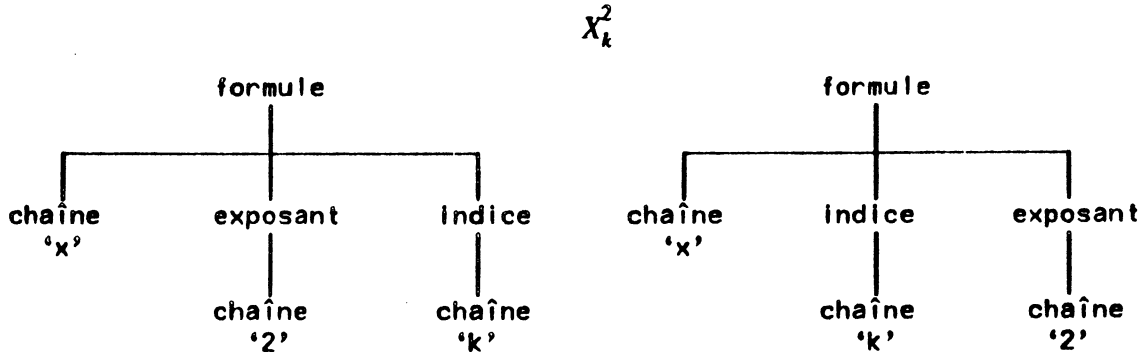


Figure 2.2 : deux représentations internes équivalentes.

Dans la figure 2.1 on peut remarquer que l'arbre n'est pas strictement conforme à ces règles : le bloc ne contient qu'un constituant puisque les symboles ouvrant et fermant n'apparaissent pas en tant que nœuds. Par souci de simplification de la représentation interne, ces deux constituants ne sont pas présents dans l'arbre. Comme ils ont toujours la même structure, celle d'un simple symbole, ils sont représentés comme des attributs du nœud bloc et une seule expression, le contenu du bloc, est donc attachée au nœud bloc. Le bloc est la seule exception de ce type.

On a vu que chaque construction avait un certain nombre de constituants spécifiques. Selon les constructions, certains sont obligatoires, d'autres sont optionnels. Dans toutes les constructions à un seul constituant (exposant, indice, racine), le constituant unique est obligatoire : s'il ne contient rien, à quoi sert un exposant (ou un indice, ou une racine) ? Dans la fraction, le numérateur et le dénominateur sont tous deux obligatoires : qu'est-ce qu'une fraction sans numérateur ou sans dénominateur ? Dans l'intégrale, les deux limites sont facultatives : on peut ainsi représenter des intégrales indéterminées. Dans la construction triple, le constituant principal est obligatoire, ainsi que l'un des constituants inférieur ou supérieur : s'il n'y a que le constituant principal, la construction triple se réduit à l'expression ou au symbole de ce constituant. De même, le vecteur doit avoir au moins deux éléments, sinon ce n'est plus un vecteur. Le bloc, enfin, a toujours ses trois composants, l'un des symboles pouvant être blanc, comme dans la formule (5).

La structure d'une formule est toujours construite par Edimath en respectant ces contraintes sur les composants obligatoires. Cependant, l'arbre d'une formule est une structure construite dynamiquement et les contraintes ne peuvent pas être respectées de façon permanente : pendant la construction de l'arbre, il existe des étapes incohérentes, de

même que après l'exécution de certaines commandes de supression. La structure des formules n'est vérifiée par Edimath qu'à certains moments où la formule doit être cohérente. Alors, en cas d'incohérence, Edimath modifie la formule en ajoutant ou en supprimant des éléments de la représentation interne.

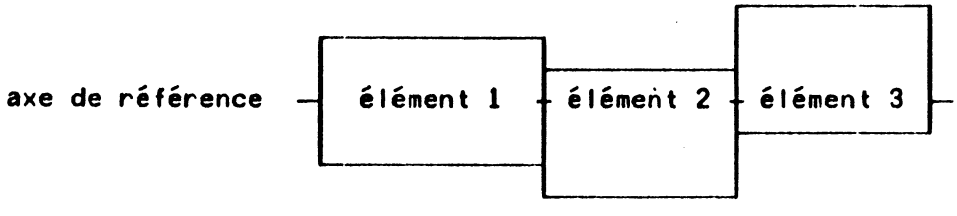
3.2. La représentation graphique

Pour afficher une formule sur un écran, ou l'imprimer sur le papier, il faut d'abord calculer la position et la dimension de chacun de ses éléments. Ce calcul se fonde sur la notion de *boîte*. A chaque construction et à chaque constituant, on fait correspondre le rectangle qui englobe son image sur l'écran ou sur le papier. Cela s'applique également aux chaînes de caractères et aux symboles, c'est à dire à tous les nœuds de l'arbre de la représentation interne, ainsi qu'à ses feuilles. TEX utilise le même principe pour formater les formules.

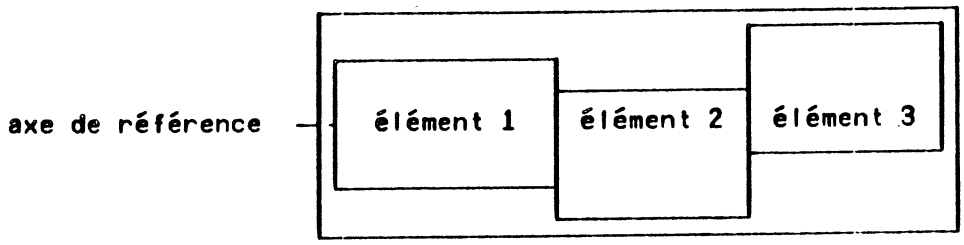
Pour chacune des huit constructions nous avons défini des *règles de présentation* qui spécifient comment les boîtes des différents constituants doivent se positionner et se dimensionner les unes par rapport aux autres, et quel corps il faut utiliser pour afficher les caractères qu'elles contiennent. En appliquant ces règles, on peut calculer la position et la dimension de chaque boîte, puis afficher ou imprimer la formule.

Chaque boîte est repérée par ses quatre côtés et son axe de référence, qui est une ligne horizontale permettant d'aligner les boîtes. L'axe de référence d'une chaîne de caractères est sa ligne de base. Celui des autres constructions est défini par les règles de présentation. Trois règles générales s'appliquent aux boîtes et permettent de les dimensionner et de les positionner en fonction de la place, dans la structure arborescente, de l'élément qu'elles représentent.

Règle 1 : Sauf indication contraire dans les règles spécifiques aux constructions, les boîtes correspondant à des éléments successifs de même niveau dans l'arbre sont alignées les unes à la suite des autres, sur leurs axes de référence.



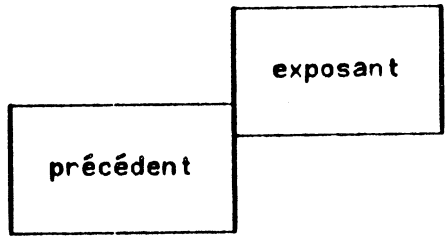
Règle 2 : La boîte d'un élément de la structure arborescente englobe les boîtes des éléments de son sous-arbre et positionne son axe de référence sur celui de la première boîte englobée. Il y a des exceptions à la règle d'héritage de l'axe de référence, mais pas à la règle d'englobement.



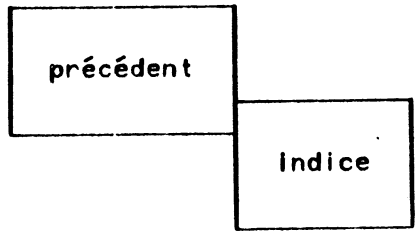
Règle 3 : Sauf exception indiquée dans les règle spécifiques, chaque boîte prend le même corps que la boîte englobante. L'attribut corps attaché à une boîte détermine le corps des caractères contenus dans la boîte.

Les règles de présentation spécifiques aux constructions définissent les exceptions aux trois règles générales. Ces règles spécifiques sont les suivantes :

exposant La boîte de l'exposant se place à la droite de la boîte de l'expression sœur précédente dans l'arbre de la représentation interne (par la suite on dira plus simplement "la boîte précédente"), le côté inférieur de la boîte exposant étant aligné avec le côté supérieur de la boîte précédente, et avec un léger décalage vers le bas si l'écran ou l'imprimante le permet. Si la boîte précédente est un indice, ce positionnement se fait par rapport à la boîte précédant l'indice. Le corps de la boîte exposant est plus faible que celui de la boîte englobante.

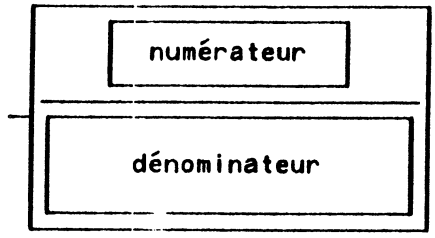


indice Les règles de présentation de l'indice sont similaires à celle de l'exposant ; seule la règle indiquant la position verticale diffère : le côté supérieur de l'indice se place en bas de la boîte précédente. Le corps de la boîte indice est plus faible que celui de la boîte englobante.

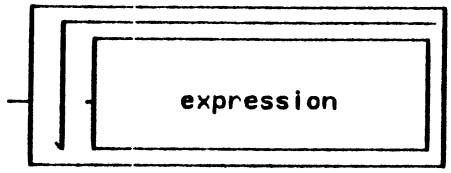


fraction La boîte du numérateur est positionnée au-dessus de la boîte du dénominateur et un espace est laissé entre les deux boîtes, pour la barre de fraction. Les deux boîtes sont centrées l'une par rapport à l'autre. L'axe de référence de la boîte de la fraction est aligné avec le haut de la boîte dénominateur. Un trait horizontal

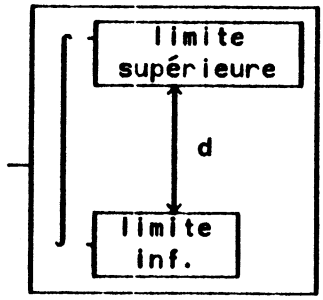
est tracé, pour la barre de fraction, entre les deux boîtes, de la longueur de la boîte fraction.



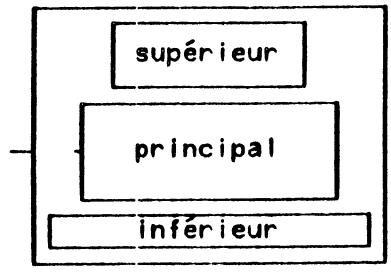
racine La boîte racine est construite à partir de la boîte de son unique constituant. Elle a le même axe de référence et laisse de l'espace à gauche et en haut pour le tracé du symbole radical $\sqrt{\ }$.



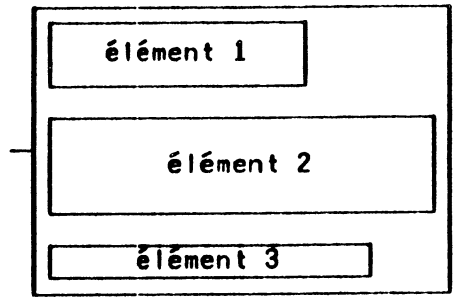
intégrale La boîte intégrale prend l'axe de référence du symbole $\int$ qu'elle contient. La distance d entre le bas de la limite supérieure et le haut de la limite inférieure est fixe. La largeur de la boîte permet le tracé d'un symbole $\int$ dont le haut et le bas sont au niveau des axes de référence des limites. Le corps des boîtes limites est plus faible que celui de la boîte intégrale.



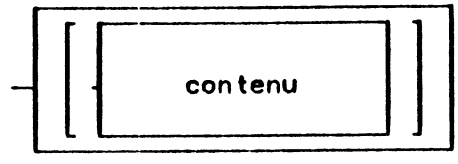
triple L'axe de référence d'une boîte triple est celui du constituant principal. Les boîtes des constituants inférieur et supérieur se placent au-dessous et au-dessus de celle du constituant principal, et les trois boîtes sont centrées verticalement. Lorsqu'un des trois constituants contient seulement un symbole élastique horizontalement (← → −), la boîte de ce constituant prend la largeur de la boîte triple (elle-même déterminée par le constituant le plus large de la construction triple), et le symbole élastique prend la longueur de sa boîte. Le corps des boîtes inférieure et supérieure est plus faible que celui de la boîte triple.



vecteur Les boîtes des différents éléments qui constituent le vecteur sont mises les unes au-dessous des autres, avec un espace entre les boîtes successives. Trois modes d'alignements des éléments sont offerts. Le mode d'alignement est un attribut du vecteur. Il permet de centrer les éléments, de les aligner sur leur côté gauche ou de les aligner sur leur côté droit. Quel que soit le mode d'alignement, le vecteur est centré verticalement, c'est à dire que son axe de référence est placé en son milieu, indépendamment des axes de référence de ses éléments. Le schéma suivant représente un vecteur à trois éléments, avec le mode d'alignement à droite.



bloc L'axe de référence de la boîte bloc est le même que celui de la boîte de l'expression qu'il contient. La hauteur de la boîte bloc est également celle de cette boîte contenue. La boîte bloc laisse de l'espace à droite et à gauche pour le tracé des symboles ouvrant et fermant qui prennent la hauteur de la boîte bloc.



Grâce aux règles de présentation générales et aux règles spécifiques, il est possible de construire, à partir de la structure arborescente, l'ensemble des boîtes qui définissent l'image d'une formule. La figure 2.3 montre l'ensemble des boîtes correspondant à la formule (1).

Les règles de présentation ont été présentées ici d'une façon générale, indépendamment de l'appareil utilisé pour l'affichage. Selon les capacités de l'appareil, ces règles sont appliquées d'une façon plus ou moins précise. Avec un terminal alphanumérique ordinaire,

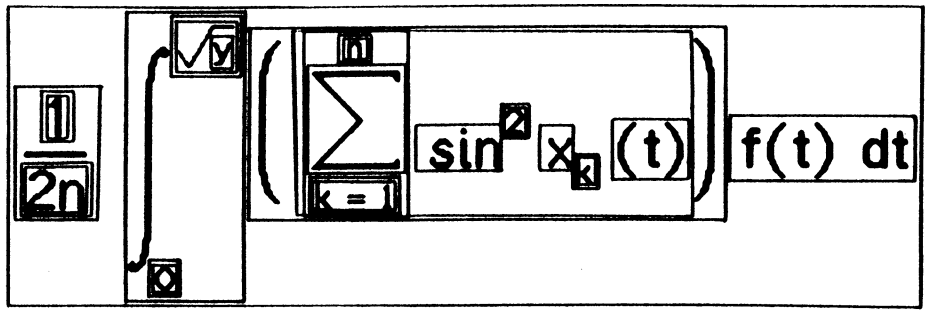


Figure 2.3 : Les boîtes d'une formule.

il n'est par exemple pas possible de changer le corps des caractères, ni de positionner les indices et les exposants d'une façon très fine. De même les centrages sont approximatifs, puisque les caractères doivent être affichés dans la grille de lignes et de colonnes de l'écran. Sur un écran graphique, une image plus précise peut être obtenue, avec plusieurs corps et des positionnements plus fins.

3.3. La représentation externe

Puisque l'image d'une formule peut être construite automatiquement à partir de sa structure logique, la meilleure façon de stocker ou de transmettre une formule est d'utiliser cette structure logique. Il suffit alors d'appliquer les règles de présentation pour en obtenir l'image sur un écran ou une feuille de papier. L'émetteur n'a pas ainsi à se préoccuper des caractéristiques de l'appareil d'affichage du destinataire.

La représentation interne des formules, qui est un arbre, pose quelques difficultés lorsqu'il s'agit de la transmettre ou même de la stocker en mémoire secondaire. Pour obtenir la *représentation externe*, stockable ou communicable, il faut la mettre sous une forme linéaire. On a retenu pour cela une notation préfixée où chaque nœud de l'arbre est représenté par une lettre indiquant son type, éventuellement suivie d'une ou deux lettres pour ses attributs (si le type en prévoit) et de tout son sous-arbre, délimité par des parenthèses. Dans cette notation, les représentations des nœuds frères sont simplement concaténées, dans le même ordre que dans l'arbre.

Cette notation est plus précisément décrite par une grammaire qui définit à la fois la syntaxe de la représentation externe et les règles de construction de l'arbre interne. La grammaire indique notamment les constituants de chaque construction, ainsi que ses attributs. Elle indique également les constituants obligatoires et optionnels.

La grammaire de la représentation externe donnée ci-dessous est décrite à l'aide du méta-langage M, dérivé de la forme de Backus-Naur (BNF).

Dans ce méta-langage (une définitiom formelle en est donnée dans l'annexe 1), chaque règle de la grammaire du langage défini est constituée d'un symbole de la grammaire suivi du signe égal et de la partie droite de la règle. Le signe égal joue le même rôle que le traditionnel ::= de la BNF : il indique que la partie droite définit le symbole de la partie gauche. Dans la partie droite,

la concaténation est simplement marquée par la juxtaposition des symboles ;
les chaînes de caractères entre apostrophes (') représentent des symboles terminaux ;
les parties entre les crochets '[' et ']' sont optionnelles ;
les parties entre '<' et '>' peuvent être répétées plusieurs fois ou omises ;
la barre oblique '/' indique une alternative ;
le point '.' marque la fin d'une règle ;
le texte entre accolades est un simple commentaire.

On verra plus loin d'autres langages définis avec le méta-langage M. Pour ces langages, on définit dans M les notions d'identificateurs, de chaînes et d'entiers. Bien que ces notions ne soient pas utiles pour la définition de la représentation externe d'Edimath, elles sont présentées ici, de façon à donner une définition complète de M.

NAME représente un identificateur, c'est à dire une suite de lettres majuscules ou minuscules, de chiffres et de '_', commençant par une lettre. On accepte comme une lettre la suite de caractères \nn où les 'n' représentent le code de la lettre en octal. On peut ainsi utiliser dans les identificateurs des caractères non-standard, notamment pour les lettres accentuées du français.

STRING représente une chaîne. C'est une suite de caractères quelconques délimités par des apostrophes. Si une apostrophe doit figurer dans la chaîne de caractères, elle est doublée. Comme pour les identificateurs, les chaînes peuvent comprendre des caractères représentés par leur code octal (après un '\'). Si un caractère '\' doit figurer dans une chaîne, il est doublé.

NUMBER représente un entier positif ou nul, sans signe, c'est à dire une suite de chiffres décimaux.

Une convention supplémentaire est utilisée ici, qui ne fait pas partie du méta-langage M : les symboles terminaux '(' et ')' qui délimitent le contenu de chaque expression sont en fait des caractères non imprimables, ce qui évite qu'ils soient confondus avec les caractères parenthèses qui apparaissent dans les chaînes de caractères de la formule. Ici, pour plus de clarté, ils ont été remplacés par des caractères imprimables.

```
Formule = 'M(' Expression ')' .
```
{ M représente la totalité de la formule (Mathématiques) }
```
Expression = Construction < Construction > .
```
{ Une expression est une suite de constructions }
```
Construction = Chaîne /
        Construction  Exposant / Construction  Indice /
        Construction  Exposant  Indice /
        Construction  Indice  Exposant /
        Fraction / Racine / Intégrale / Triple / Vecteur / Bloc .
```
{ Les huit constructions d'Edimath, plus la chaîne de caractères }
```
Chaîne = 'C' Alphabet '(' Texte ')' .
```
{ "Alphabet" est un attribut de la construction Chaîne }
```
Alphabet = '1' / '2' / '3' / '4' .
```
{ chaque alphabet est identifié par un chiffre. 1 : ASCII,
2 : mathématique, 3 : grec... }
```
Texte  = Caractère < Caractère > .
```
{ Une chaîne contient une suite de caractères }
```
Exposant = 'P(' Expression ')' .
```
{ P identifie un exposant (Power) }
```
Indice = 'S(' Expression ')' .
```
{ S identifie un indice (Subscript) }
```
Fraction = 'F(N(' Expression ')D(' Expression '))' .
```
{ F=Fraction, N=Numérateur, D=Dénominateur }
```
Racine = 'R(' Expression ')' .
```
{ R identifie une racine }
```
Intégrale = 'I' / 'I(L(' Expression '))' / 'I(U(' Expression '))' /
        'I(L(' Expression ')U(' Expression '))' .
```
{ I=Intégrale, L=limite inférieure (Lower),
U=limite supérieure (Upper) }
```
Triple = 'T(t(' ExpSymb ')l(' ExpSymb ')u(' ExpSymb '))' /
        'T(t(' ExpSymb ')l(' ExpSymb '))' /
        'T(t(' ExpSymb ')u(' ExpSymb '))' .
```
{ T=Triple, t=composant principal de triple, l=composant
inférieur (lower), u=composant supérieur (upper) }
```
ExpSymb = Expression / 'W(' SymbHoriz ')' / 'H(' SymbVert ')' .
```
{ W=symbole élastique horizontalement, H=symbole de grande taille }
```
SymbHoriz = '>' / '<' / '-' .
```
{ > pour →, < pour ←, - pour − }
```
SymbVert = 'S' / 'P' / 'U' / 'I' .
```
{ S pour Σ, P pour Π, U pour ∪, I pour ∩ }

```
Vecteur = 'V' ModeAlign '(E(' Expression ')E(' Expression ')'
        < 'E(' Expression ')' > ')' .
        { V=Vecteur, modealign=attribut d'alignement des éléments,
        E=Elément }
ModeAlign = 'C' / 'L' / 'R' .
        { C=Centré, L=aligné à gauche (Left), R=aligné à droite (Right) }
Bloc   = 'B' Parenth Parenth '(' Expression ')' .
        { B=Bloc, parenth=attributs symboles ouvrant et fermant }
Parenth = '(' / ')' / '{' / '}' / '[' / ']' / '|' / ' ' .
```

A titre d'exemple, la formule (1) s'écrit dans la représentation externe :

$$\frac{1}{2n}\int_0^{\sqrt{y}}\left(\sum_{k=1}^{n}\sin^2 x_k(t)\right) f(t)\,dt$$

```
M(                                      { toute la formule }
  F(N(Cl(1))D(Cl(2n)))                  { la fraction }
  I(L(Cl(0))U(R(Cl(y))))                { l'intégrale avec ses deux limites }
  B[](                                  { la grande parenthèse }
      T(t(H(S))l(Cl(k=1))u(Cl(n)))      { la sommation }
      Cl(sin)                           { la fonction sinus }
      P(Cl(2))                          { l'exposant 2 après le sinus }
      Cl(x)                             { la variable x }
      S(Cl(k))                          { l'indice k }
      Cl([t])                           { la variable (t) }
     )                                  { fin de la parenthèse }
  Cl(f[t]dt)                            { la chaîne après la parenthèse }
 )                                      { fin de la formule }
```

Les caractères '(' et ')' sont les délimiteurs de la structure (normalement non imprimables) et les caractères '[' et '|' représentent les caractères imprimables parenthèses qui apparaissent dans l'image de la formule. La forme externe est linéaire. Elle a été ici découpée en lignes pour plus de clarté.

4. L'INTERFACE UTILISATEUR D'EDIMATH

Avec la structure des formules telle qu'elle vient d'être présentée, la création de toute expression mathématique devient simple pour l'utilisateur. Celui-ci n'a qu'à décrire la structure de l'expression dans des termes qui lui sont naturels (c'était l'un des objectifs de la définition de la structure) et l'éditeur affiche immédiatement, chaque fois qu'un nouvel élément est ajouté, la forme graphique de la formule, dans l'état où elle se trouve. Cela permet notamment à l'utilisateur de vérifier qu'il obtient bien ce qu'il souhaitait, ce qui est particulièrement important pour les formules où l'imbrication des expressions rend délicat un parenthésage correct ; les utilisateurs de TEX ou de Eqn en ont l'expérience.

4.1. La saisie d'une formule

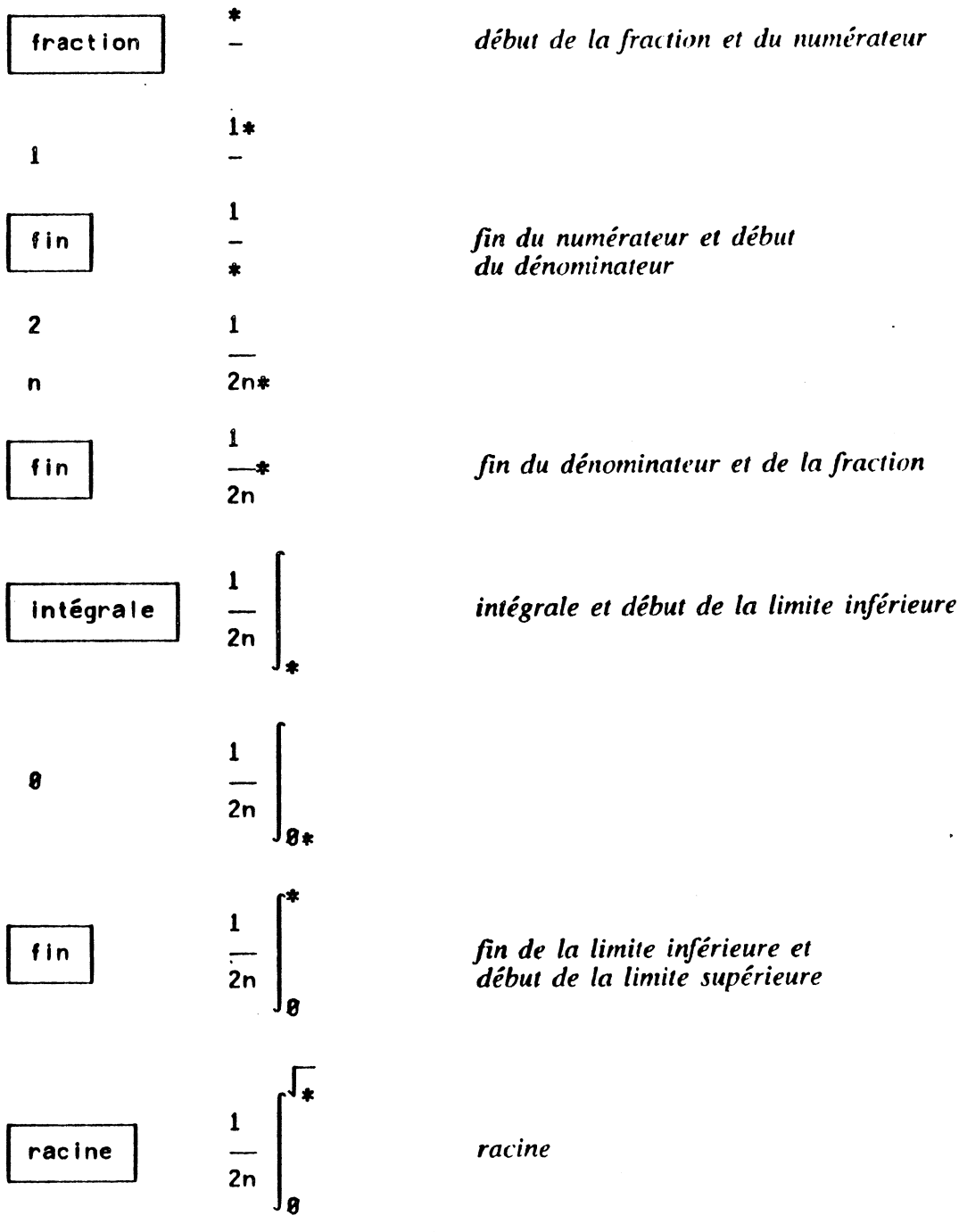
La création d'une formule se fait essentiellement au clavier. Les caractères qui composent la formule sont entrés de façon conventionnelle et les constructions peuvent être indiquées soit à l'aide de touches de fonction, soit à l'aide de menus. Les menus sont disponibles sur certaines versions d'Edimath, mais lors de la saisie ils ne présentent en fait d'intérêt que pour les utilisateurs peu entraînés. Ils sont surtout utilisés dans les opérations de modification.

Une touche de fonction (et éventuellement une entrée dans un menu) est associée à chacune des huit constructions. La saisie d'une formule se fait alors d'une façon très proche de la façon dont on la dicterait. Pour lever les ambiguïtés de la forme orale, une touche de fonction supplémentaire permet d'indiquer la fin de chaque construction ou constituant. Comme les constituants sont directement liés aux constructions, il n'est pas nécessaire de les indiquer explicitement ; chaque fois qu'une construction nouvelle est entrée, l'éditeur crée les constituants de cette construction.

On a vu que certaines constructions avaient des attributs (symboles encadrant les blocs, mode de centrage des vecteurs, alphabets des chaînes de caractères). Lorsque l'utilisateur frappe la touche de fonction d'une construction possédant des attributs, l'éditeur demande explicitement la valeur des attributs en affichant un message avec la liste des valeurs possibles. L'utilisateur indique la valeur retenue en frappant une seule touche alpha-numérique. La valeur de l'attribut doit normalement être indiquée dès que la nouvelle construction est créée, mais pour rendre les choses plus simples, il y a deux exceptions à cette règle : les alphabets et les symboles fermants de blocs. L'attribut "symbole fermant" des blocs est demandé lorsque l'utilisateur termine un bloc en frappant la touche de fin d'expression. L'attribut "alphabet" n'est pas demandé à chaque nouvelle chaîne. D'ailleurs les chaînes de caractères ne sont pas créées comme le sont les constructions, par une touche de fonction, mais dès qu'une touche alpha-numérique est

frappée. La chaîne ainsi créée prend comme attribut "alphabet" l'alphabet courant et celui-ci peut être changé à tout moment grâce à une touche de fonction (ou un menu).

L'exemple ci-dessous montre comment la formule (1) est saisie sous Edimath. La colonne de gauche donne la séquence des touches frappées par l'utilisateur (les touches de fonction sont encadrées). La colonne du milieu présente l'image affichée sur un écran alphanumérique (le curseur est représenté par *). La colonne de droite est un simple commentaire.



y

$$\frac{1}{2n}\int_{0}^{\sqrt{y*}}$$

fin

$$\frac{1}{2n}\int_{0}^{\sqrt{y*}}$$

fin de la racine

fin

$$\frac{1}{2n}\int_{0}^{\sqrt{y}}*$$

fin de la limite supérieure et fin de l'intégrale

bloc

$$\frac{1}{2n}\int_{0}^{\sqrt{y}}(*$$

début de bloc avec

(

symbole ouvrant '('

triple

début de triple

symbole

$$\frac{1}{2n}\int_{0}^{\sqrt{y}}\left[\sum_{*}\right.$$

constituant principal : symbole Σ

S

k
=
1

fin

$$\frac{1}{2n}\int_{0}^{\sqrt{y}}\left[*\sum_{k=1}\right.$$

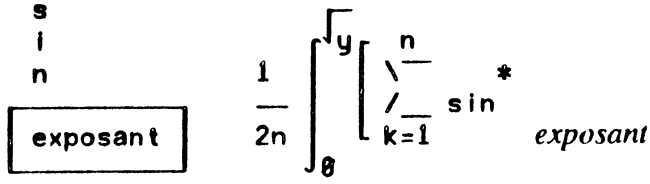
fin d'expression inférieure et début d'expression supérieure

n

fin

$$\frac{1}{2n}\int_{0}^{\sqrt{y}}\left[\sum_{k=1}^{n}\right.*$$

fin d'expression supérieure et fin de triple



etc...

4.2. Les modifications de formules

A tout moment lors de la création d'une formule, il est possible de revenir en arrière pour opérer une modification. Evidemment, les modifications sont également possibles sur une formule dont la saisie est terminée.

Si pour le texte la notion d'insertion est claire, il en va différemment pour les formules. Il ne suffit pas de positionner une marque d'insertion sur l'écran pour que les caractères frappés ensuite se placent automatiquement à l'endroit voulu dans la formule. L'exemple suivant le montre :

$$\sqrt{a+b}|-1 \longrightarrow \begin{cases} \sqrt{a+b+c}-1 & (a) \\ \sqrt{a+b}+c-1 & (b) \end{cases} \qquad (6)$$

Si le curseur est à la position marquée par une barre verticale dans la formule de gauche, la frappe des caractères '+c' peut donner aussi bien le résultat (a) que le résultat (b). Ce type d'ambiguïté est très courant dans les formules. Pour cette raison Edimath demande qu'une partie de formule soit sélectionnée avant toute insertion. L'insertion peut alors avoir lieu, par une commande explicite indiquant où, par rapport à la partie sélectionnée, doivent se placer les nouvelles expressions qui vont être entrées.

La sélection se fait également d'une façon particulière pour les formules. Là encore, la nature bidimensionnelle de l'écriture mathématique s'oppose à la linéarité du texte. Avec le texte, il suffit de désigner deux points sur l'écran pour définir sans ambiguïté une partie d'un document sur laquelle on veut appliquer une commande : les deux points marquent le début et la fin de cette partie. Dans une formule, cela n'est applicable que si une structure sous-jacente, comme celle d'Edimath, définit un ordre entre les éléments. C'est pour cette raison que la sélection se fait à l'aide de la structure.

L'utilisateur dispose de deux commandes : "Sélection suivant" et "Sélection englobant". L'action de ces commandes diffère légèrement suivant qu'il y a déjà une sélection ou non. Lorsque rien n'est sélectionné, la commande "Sélection suivant" sélectionne le caractère sur lequel se trouve le curseur et la commande "Sélection englobant" sélectionne la plus petite boîte contenant la position courante du curseur. Dès qu'un élément au moins,

même un simple caractère, est sélectionné, la commande "Sélection suivant" ajoute à la sélection courante l'objet (caractère, construction ou constituant) suivant le dernier objet sélectionné, l'ordre des objets étant indiqué par la structure de l'arbre. La commande "Sélection englobant", elle, remplace la sélection courante par la boîte correspondant au premier nœud de l'arbre dont tous les éléments de la sélection courante sont des descendants. On voit que, pour le confort de l'utilisateur, la sélection considère la structure à un niveau plus fin que dans l'arbre, puisqu'elle permet de sélectionner des caractères séparément, et pas seulement des chaînes complètes.

On a présenté là le processus de sélection en termes de structure, mais l'utilisateur peut très bien le considérer d'un point de vue purement graphique et, en utilisant les deux commandes de sélection, se guider uniquement sur l'image qu'il voit sur l'écran. Pour faciliter cette façon de faire, l'éditeur met en évidence sur l'écran la sélection courante dès qu'elle est modifiée. D'ailleurs, sur le Macintosh, il est possible de sélectionner directement, avec la souris, simplement en balayant la partie de l'image que l'on souhaite sélectionner. Dans ce cas, le résultat obtenu est le même que celui obtenu par les commandes "Sélection suivant" et "Sélection englobant".

Une fois qu'une partie de formule est sélectionnée, l'insertion peut se faire à l'aide de deux commandes différentes : "Insérer devant" et "Insérer après", permettant ainsi de lever des ambiguïtés telles que celle montrée sur la formule (6). Pour obtenir le résultat (a), on sélectionne le caractère 'b' et on insère après. Pour obtenir le résultat (b), on peut soit sélectionner toute la racine et insérer après, soit sélectionner '−' et insérer avant. Dès qu'une commande d'insertion est appelée, la formule est réaffichée avec une zone blanche à l'endroit de l'insertion. L'utilisateur entre alors l'expression à insérer exactement comme il crée une formule et il voit la formule se redessiner en fonction des ajouts qu'il fait.

Deux autres commandes peuvent être appliquées à une partie sélectionnée : "Supprimer" et "Remplacer". La commande "Supprimer" réaffiche la formule, après avoir supprimé la partie sélectionnée. La commande "Remplacer" supprime également la partie sélectionnée mais réaffiche la formule avec une zone blanche à la place de la sélection, permettant ainsi à l'utilisateur d'entrer n'importe quelle expression à cet endroit, comme il le fait après une commande d'insertion.

Comme on rencontre souvent dans les développements mathématiques des formules successives faisant apparaître des expressions identiques, Edimath fournit un moyen de construire rapidement de nouvelles formules à partir d'éléments de formules traitées précédemment. Une fois qu'une partie de formule est sélectionnée, l'utilisateur peut sauver cette partie dans l'un des "registres" à sa disposition. Il peut ainsi conserver plusieurs éléments de formules. Pendant la création ou l'insertion, il peut à tout moment, par une simple commande, insérer le contenu de l'un de ces registres à la position courante du curseur. Notons que c'est la structure des éléments de formules, et non leur image

graphique, qui est stockée dans les registres. Ainsi, il est possible de les insérer en tout point d'une formule puisque leur image est recréée en fonction de leur nouvelle position dans la structure.

C'est d'ailleurs une des propriétés importantes d'Edimath de prendre en charge le dessin des formules à partir des indications de structure fournies par l'utilisateur. Comme on l'a vu sur le scénario de saisie montré plus haut, les différents symboles se dimensionnent automatiquement (barre de fraction, radical, parenthèses, etc...). De même Edimath choisit le corps des caractères (ce n'est pas visible sur le scénario, puisque l'image montrée est celle qu'on obtient sur un terminal alpha-numérique standard). Sur les machines qui le permettent, Edimath offre en plus la possibilité d'agrandir ou de réduire la taille des formules qu'il traite. Cela permet notamment d'éditer des formules importantes, en choisissant une petite taille, ou de les rendre plus lisibles, en choisissant une grande taille. Ces changements de taille ne se font pas par transformation géométrique de l'image mais par calcul à partir de la structure, en utilisant des caractères de corps plus ou moins fort.

5. LA MISE EN OEUVRE D'EDIMATH

La mise en œuvre d'Edimath a déjà été présentée à plusieurs reprises [Quint 82, 83, 84]. Elle n'est pas détaillée ici. L'accent est plutôt mis sur les particularités d'Edimath vis à vis des autres éditeurs et sur les problèmes originaux qui se sont posés lors de la réalisation. Quelques détails techniques se trouvent dans l'annexe 5.

5.1. L'architecture

Edimath se compose de deux programmes indépendants. L'un est l'**éditeur** proprement dit, l'autre est le **formateur**. L'éditeur a la charge de construire et de modifier les formules en mode interactif sur l'écran, le formateur de les sortir sur papier.

L'éditeur produit des fichiers qui contiennent la représentation externe des formules créées par l'utilisateur. Il peut aussi lire ces fichiers pour permettre à l'utilisateur de modifier les formules existantes. Ce sont ces mêmes fichiers que le formateur lit pour créer l'image des formules sur papier.

Lors de la création d'une formule, l'éditeur construit la représentation externe en suivant la grammaire de cette représentation. C'est la grammaire qui lui indique les constituants à créer chaque fois qu'une nouvelle construction est entrée par l'utilisateur. L'éditeur crée toujours tous les constituants possibles, même lorsqu'ils sont optionnels. En même temps que de nouveaux éléments sont créés dans la représentation externe, les boîtes

correspondantes sont construites. Elles sont dimensionnées et positionnées grâce aux règles de présentation, puis affichées. L'écran montre ainsi l'image de la formule construite au fur et à mesure de la création. De plus, l'utilisateur est guidé par l'éditeur qui crée pour lui tous les composants possibles ; il n'a qu'à fournir le contenu de ces composants ou à supprimer ceux dont il n'a pas besoin.

Le formateur lit les fichiers produits par l'éditeur. A partir de la représentation externe contenue dans ces fichiers, il construit les boîtes, en appliquant les règles de présentation adaptées à l'imprimante. Ce traitement est le même que celui qui est fait par l'éditeur lorsqu'il lit une formule contenue dans un fichier, à l'exception de quelques détails concernant les règles de présentation, qui peuvent différer légèrement pour un écran et pour une imprimante. Mais une autre différence avec l'éditeur est que le formateur analyse le contenu des chaînes de caractères. Il recherche les noms de fonctions mathématiques et les chiffres, qu'il imprime en caractères romains, alors que les autres caractères sont imprimés en italique. Il cherche également les principaux symboles opératoires de façon à les espacer d'avantage que les autres caractères. Ce traitement du formateur évite à l'utilisateur de se préoccuper des détails tels que le style des caractères ou les espacements. De plus, même si l'éditeur demandait à l'utilisateur d'entrer ces indications, il serait incapable de visualiser clairement ces détails sur la plupart des écrans. Dans Edimath l'idée est bien de demander à l'utilisateur de décrire la structure *logique* des formules, le système prenant en charge tout l'aspect *physique*.

5.2. Les principes de fonctionnement

Dans l'éditeur, l'arbre de la représentation interne n'existe pas en tant que tel en mémoire. Une seule structure de données décrit à la fois la représentation interne et la représentation graphique : la liste de boîtes. Il s'agit d'une liste ordonnée d'éléments (les boîtes) qui contiennent chacun

- les informations graphiques de la boîte : position et dimensions sur l'écran, corps des caractères contenus,
- les informations structurelles : type d'élément et niveau dans l'arbre (virtuel) de la représentation interne.

La liste est ordonnée de façon qu'il soit possible de retrouver la position structurelle de chaque boîte à partir de sa position dans la liste et de son niveau (indiqué dans la boîte). L'exemple de la figure 2.4 montre les relations entre la structure arborescente et l'ordre des boîtes dans la liste.

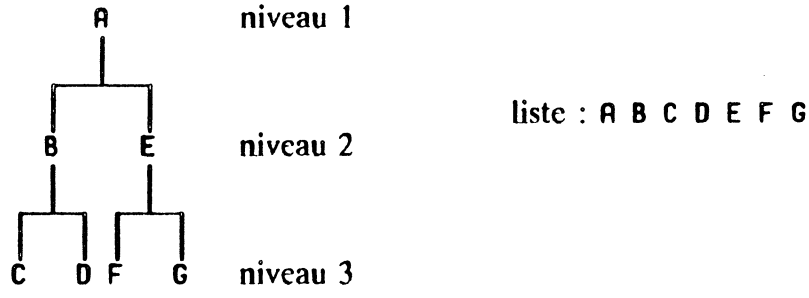


Figure 2.4 : l'arbre et la liste des boîtes.

Cette structure de liste est doublée d'une pile qui contient le même type d'éléments et dans laquelle se trouvent toutes les boîtes correspondant aux nœuds qui sont sur le chemin depuis la racine de l'arbre jusqu'au nœud courant (l'élément que l'utilisateur doit entrer). Ces boîtes sont dupliquées dans les deux structures. Les boîtes de la pile portent un indice donnant le rang de leur homologue dans la liste et les boîtes de la liste portent l'indication du niveau de leur homologue dans la pile.

A titre d'exemple, avec l'arbre de la figure 2.4, si le nœud courant est `F`, la pile contient, du fond vers le sommet, les boîtes `A E F`. Ces deux structures (liste et pile) sont maintenues cohérentes pendant toutes les opérations d'édition. Elles permettent à la fois des traitements efficaces et une représentation compacte des formules (il n'y a pas les pointeurs habituels d'un arbre).

Pendant la création d'une formule ou d'une expression insérée dans une formule existante, la boîte courante est la dernière créée qui n'est pas encore terminée. La pile contient ainsi toutes les boîtes englobant la boîte en cours de construction et permet immédiatement de redimensionner ces boîtes en fonction des ajouts de l'utilisateur. Dès qu'une boîte de la pile change de taille, la liste permet de retrouver rapidement toutes les boîtes qui doivent être décalées : ce sont toutes celles qui suivent dans la liste la boîte modifiée et qui sont de niveau supérieur ou égal.

Lors des modifications, l'utilisateur travaille essentiellement sur la représentation graphique. L'éditeur doit faire le lien entre la forme graphique et la structure interne. Là encore, les structures de données retenues facilitent les opérations. Pendant la sélection, la première boîte sélectionnée se trouve simplement en cherchant, depuis la fin de la liste vers le début, la première boîte contenant la position courante du curseur. L'extension de la sélection se fait également de façon triviale sur la liste.

Le principe qui a été retenu pour la sélection, avec ces deux commandes, permet à l'utilisateur de sélectionner n'importe quelle partie de formule et assure que cette partie est cohérente avec la structure. Ainsi les commandes d'édition ("Insérer avant", "Insérer

après", "Supprimer" ou "Remplacer") ne nécessitent pas de longues vérifications pour assurer que la nouvelle formule produite aura une structure valide.

Certaines vérifications sont néanmoins nécessaires, qui permettent une certaine souplesse dans le processus de sélection, du point de vue de l'utilisateur. Prenons à titre d'exemple la fraction qui apparaît au début de la formule (1). Chacun de ses constituants, numérateur et dénominateur, contient une seule construction, une chaîne de caractères. La boîte du constituant et celle de son contenu (la chaîne de caractères) se superposent donc exactement sur l'écran, bien que, dans l'arbre, elles soient à des niveaux différents. Il n'est donc pas possible à l'utilisateur de distinguer sur l'écran entre la sélection du dénominateur et la sélection de la chaîne qu'il contient, '2n'. Si l'utilisateur veut ajouter une autre expression dans le dénominateur, après la chaîne '2n', il devrait sélectionner cette chaîne et appliquer ensuite la commande "Insérer après". S'il a sélectionné la boîte dénominateur, cette même commande ne pourra pas opérer, puisque la structure ne prévoit pas d'élément suivant un dénominateur. Pour éviter de refuser une commande pour cause de sélection confuse, l'éditeur tente d'ajuster la sélection de façon que la commande soit valide. Dans le cas que nous avons pris en exemple, la sélection est automatiquement modifiée (la chaîne est sélectionnée au lieu du dénominateur) avant d'appliquer la commande d'insertion.

Un autre type de traitement effectué automatiquement par l'éditeur, est la simplification des formules. Bien que la validité de la structure soit effectivement contrôlée tout au long de la création ou de l'insertion, les suppressions peuvent conduire à des formules mal structurées. Deux stratégies complémentaires sont adoptées dans ce cas, qui permettent toutes deux de rétablir une structure correcte. L'une ajoute de nouveaux éléments, l'autre en supprime.

L'ajout a lieu lorsqu'un constituant obligatoire a été enlevé. Il s'agit dans ce cas d'une erreur de la part de l'utilisateur et il faut la lui signaler tout en lui permettant de la corriger. Reprenons l'exemple de la fraction pour illustrer cela. Si, après avoir sélectionné le dénominateur, l'utilisateur applique la commande "Supprimer", on obtient une fraction sans dénominateur, ce qui n'est pas conforme à la définition de la structure. Dans ce cas, l'éditeur maintient le numérateur et lui crée pour contenu une chaîne contenant le seul caractère '?' qui apparaît sur l'écran. L'utilisateur peut ensuite remplacer le point d'interrogation comme n'importe quel caractère ; il peut aussi sélectionner toute la fraction et la supprimer, ce qui sera accepté.

La suppression automatique a lieu lorsqu'une construction ne comporte pas le nombre minimum de constituants optionnels. Une construction triple n'est utile que si, en plus du constituant principal, il y a au moins un constituant inférieur ou supérieur. De même un vecteur n'a de sens que s'il a plusieurs éléments. Aussi, une construction triple ne comportant que le constitutant principal est simplifiée et remplacée par l'expression

contenue dans le constituant principal. Le vecteur à élément unique est remplacé par l'expression contenu dans cet unique élément.

Toutes ces interprétations réalisées par l'éditeur permettent d'échapper à la trop grande rigidité qui serait due à un respect strict des sélections et des commandes de l'utilisateur. Cette souplesse introduite dans le comportement de l'éditeur est la condition nécessaire pour le rendre réellement utilisable.

5.3. Le réaffichage

D'une manière générale, un des problèmes les plus difficiles posés par les éditeurs interactifs est celui du réaffichage. Pour résoudre ce problème, différentes stratégies ont été appliquées dans Edimath. En effet, les techniques mises en œuvre dans les éditeurs de texte [Gosling 81] ne peuvent s'appliquer directement aux formules.

Dans un premier temps, nous avons développé deux algorithmes de réaffichage différents :

- pour la création des formules (c'est à dire lorsque l'utilisateur entre les expressions constituant la formule dans l'ordre proposé par l'éditeur, sans retour en arrière),
- pour les modifications (lorsque l'utilisateur opère à l'intérieur d'une expression déjà créée).

L'idée de départ était qu'une formule changeait évidemment beaucoup au cours de sa création, mais que les modifications apportées par la suite étaient relativement rares, et qu'on pouvait donc utiliser un algorithme moins performant pour ces modifications. L'effort principal a ainsi porté sur le réaffichage effectué pendant la création (voir le scénario de saisie plus haut) alors qu'une technique très simple a été utilisée, au moins au début du projet, pour le réaffichage des modifications.

A chaque modification, (suppression, insertion, remplacement à l'intérieur d'une formule et non pas en fin de formule), la formule est complètement réaffichée : l'image de la totalité de la formule est effacée, la nouvelle image est recalculée entièrement (en construisant les boîtes) à partir de la forme externe mise à jour par la commande de modification, et cette image est enfin affichée. Cette technique simple s'est révélée satisfaisante dans de nombreux cas et notamment pour les formules peu complexes ou sur les machines où l'affichage est très rapide. En effet les temps de calcul sont pratiquement insensibles, mais sur un terminal alpha-numérique connecté à 4800 bits/s., le temps d'affichage peut atteindre plusieurs secondes pour une formule complexe. Lorsque seuls quelques caractères doivent être déplacés le comportement de l'éditeur devient alors difficilement supportable.

Pendant la création d'une formule, au contraire, le réaffichage est optimisé : seules les parties qui changent sont réaffichées. Tant que l'utilisateur suit les propositions de l'éditeur sans déplacer lui-même le curseur (voir le scénario), un réaffichage incrémental a lieu. Les règles de réaffichage sont simples si on remarque que les boîtes peuvent être classées en deux catégories selon leur comportement :

- Première catégorie : les boîtes qui s'étendent. Ce sont les boîtes dont la hauteur et/ou la largeur peuvent changer, sans que le coin supérieur gauche ne bouge. Toutes les boîtes de la pile sont dans cette catégorie.

- Deuxième catégorie : les boîtes qui sont déplacées. Ce sont les boîtes dont ni la largeur ni la hauteur ne changent, mais dont la position peut varier. Les seules boîtes de cette catégorie sont celles qui ne sont pas dans la pile (elles sont donc terminées).

Pour traiter les boîtes de la première catégorie, l'éditeur parcourt toute la pile, et pour chaque boîte à laquelle est associé un tracé, le tracé est étendu (barre pour une fraction, radical pour une racine, parenthèse ou symbole équivalent pour un bloc).

Pour traiter les boîtes de la seconde catégorie, l'éditeur travaille en deux temps. Il fait un traitement pour les déplacements horizontaux et un autre pour les déplacements verticaux. Dans le sens horizontal, les seuls décalages possibles sont ceux des constituants appartenant à des constructions qui demandent un centrage de leurs composants (triple, vecteur, fraction). L'éditeur cherche donc dans la pile les boîtes de type triple, vecteur ou fraction, et pour chacune de ces boîtes il traite toutes les boîtes contenues au premier niveau. Ces boîtes contenues au premier niveau sont recentrées dans leur boîte englobante (qui est une boîte de la première catégorie et qui a donc été élargie précédemment), et celles qui sont déplacées par le recentrage sont d'abord effacées sur l'écran, puis toutes les boîtes qu'elles contiennent sont décalées horizontalement (centrées) et finalement réaffichées à leur nouvelle position.

Dans le sens vertical, les déplacements sont plus complexes. Si on observe que les formules créées par Edimath se développent vers le bas et vers la droite, leur coin supérieur gauche restant fixe, on peut répartir les boîtes en trois catégories, selon leur comportement au cours des extensions verticales :

- Les boîtes qui peuvent s'étendre vers le haut mais pas vers le bas, l'axe de référence pouvant être déplacé : exposant, numérateur, limite supérieure d'intégrale, composant supérieur de triple.

- Les boîtes qui peuvent s'étendre vers le bas mais pas vers le haut, l'axe de référence pouvant être déplacé : indice, dénominateur, élément de vecteur, limite inférieure d'intégrale, composant inférieur de triple et la formule elle-même.

- Les boîtes qui peuvent s'étendre à la fois vers le haut et vers le bas, mais dont l'axe de référence reste fixe : racine, fraction, bloc, intégrale, triple, constituant principal de triple, vecteur.

Chaque fois qu'une nouvelle boîte est créée durant la saisie, elle est mise au sommet de la pile et au bout de la liste. Selon son type, elle occupera, dans le sens vertical, une place plus ou moins importante sur l'écran : une fraction ou une intégrale demande plus de place qu'une simple chaîne de caractères. De plus cette boîte occupe une position déterminée par rapport à la boîte précédente. Cette position est également définie par le type de la boîte. Connaissant le type de la boîte, l'éditeur peut donc calculer la taille minimum qui lui est nécessaire, c'est à dire la taille occupée par tous ses constituants obligatoires en supposant que chacun contient un unique caractère.

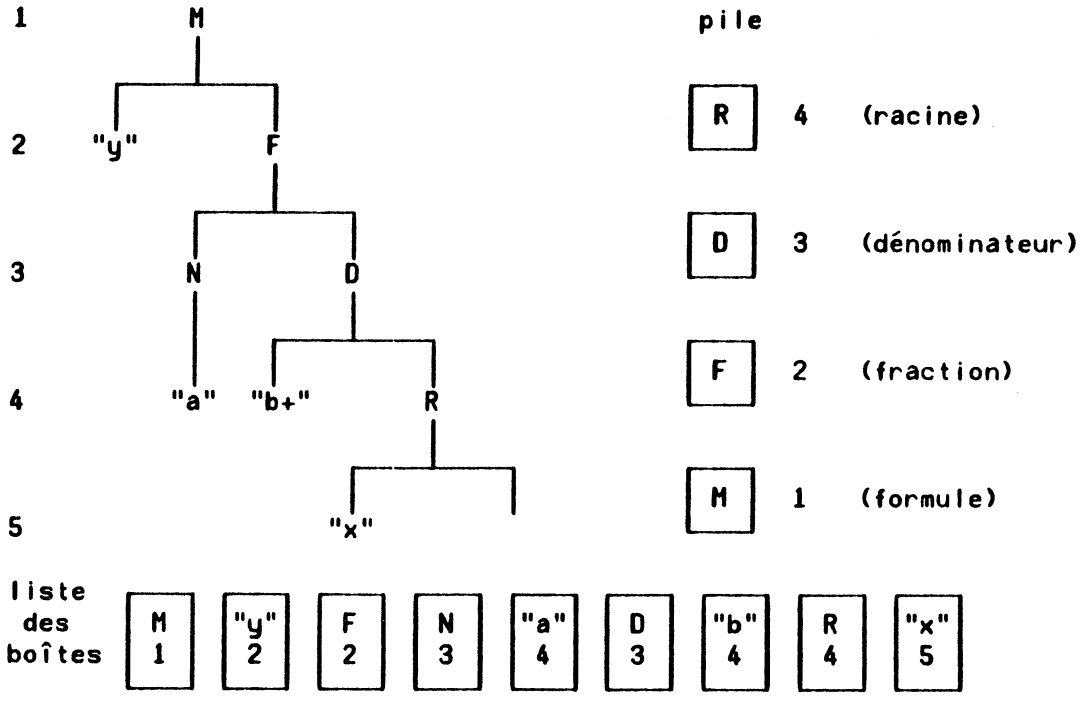
Si l'espace nécessaire n'est pas disponible dans la boîte de sommet de pile, (qui est la boîte immédiatement englobante de la nouvelle boîte), cette boîte de sommet de pile doit être étendue vers le haut et/ou vers le bas, selon l'espace manquant. Si le type de la boîte à étendre autorise cette extension, l'extension a lieu et le même traitement s'applique récursivement en descendant la pile si l'extension provoque un débordement par rapport à la boîte englobante (la suivante dans la pile). Si au contraire le type de la boîte à étendre interdit l'extension nécessaire, cette boîte est décalée en sens inverse pour libérer néanmoins l'espace voulu. Ce décalage provoque le décalage de tout son contenu. Mais il faut aussi vérifier que le décalage de la boîte elle-même ne provoque pas de débordement par rapport à sa boîte englobante, auquel cas il faudrait également étendre, de la même façon la boîte englobante.

Au cours de cet algorithme récursif, on garde la trace des boîtes de plus bas niveau (les plus "grosses") qui sont étendues ou décalées. A la fin de l'algorithme, le contenu des boîtes décalées est réaffiché (il a été effacé de l'écran avant que le décalage soit effectué). Les boîtes étendues verticalement sont réaffichées (seulement leur attribut graphique, pas leur contenu) : les symboles délimiteurs des blocs sont prolongés verticalement ainsi que le radical des racines.

On peut examiner le comportement de cet algorithme sur un exemple simple. Supposons que l'on soit en train de créer la formule suivante, où le caractère □ représente le curseur sur l'écran, et que l'on veuille entrer un exposant :

$$y = \frac{a}{b + \sqrt{x\square}} \longrightarrow y = \frac{a}{b + \sqrt{x^{2\square}}} \qquad (7)$$

Avant l'entrée de la commande "Exposant", l'arbre se présente de la façon suivante, et est représenté par la pile et la liste de boîtes ci-dessous (les chiffres indiquent le niveau de chaque boîte) :



Lorsque l'utilisateur entre la commande "Exposant", avant de créer la boîte Exposant à la suite de la boîte de la chaîne 'x', l'éditeur vérifie s'il a la place d'afficher cette nouvelle boîte. La boîte en sommet de pile, Racine, n'est pas assez haute, il faut donc l'étendre vers le haut, ce qui est possible pour une racine. Mais elle déborde en haut de la boîte suivante de la pile, celle du dénominateur, qui devrait donc également être étendue vers le haut. Un dénominateur ne peut pas être étendu vers le haut ; le côté supérieur de sa boîte ne bouge donc pas mais son côté inférieur et son axe de référence sont décalés vers le bas, ainsi que toutes les boîtes englobées, c'est à dire toutes les boîtes de la liste qui suivent la boîte dénominateur et qui sont de niveau inférieur : les boîtes 'b+', Racine et 'x'. La boîte dénominateur étant étendue vers le bas, il faut aussi étendre la boîte suivante de la pile, la boîte fraction (une fraction peut être étendue vers le bas), et de même la boîte formule. La plus grosse boîte qui a été déplacée est la boîte dénominateur, elle sera donc réaffichée, ainsi que tout son contenu (les caractères 'b+', le radical $\sqrt{}$ et le caractère 'x'). Les boîtes étendues sont les boîtes Fraction et Formule, mais comme il n'y a pas de tracé vertical associé à ces types de boîte, rien n'est réaffiché pour elles.

Pour trouver les modifications à faire dans le sens horizontal, l'éditeur parcourt la pile du sommet vers le fond et étend toutes les boîtes vers la droite. La boîte racine, en sommet de pile, a un tracé associé, le radical est donc prolongé horizontalement sur l'écran. La boîte suivante dans la pile, le dénominateur, n'a pas de tracé associé, mais la suivante, la fraction, en a un. La barre de fraction est donc prolongée. De plus La boîte fraction induit un centrage ; après l'avoir étendue, l'éditeur cherche sa boîte numérateur (c'est la

suivante dans la liste), et comme elle est plus étroite que la boîte fraction, il l'efface de l'écran, la recentre et la réaffiche. Aucun recentrage n'est nécessaire pour la boîte dénominateur, puisqu'elle a la même largeur que la boîte fraction. Au fond de la pile, la boîte formule ne nécessite aucune action particulière.

Cet algorithme permet non seulement d'évaluer les réaffichages à effectuer pendant la phase de création des formules, mais il est également utilisé pour construire l'image d'une formule depuis sa forme externe parenthésée. En effet, la construction d'une image se résume essentiellement au calcul de la dimension et de la position des boîtes de la liste. Une fois cette liste construite il reste seulement à afficher le contenu de chaque boîte, et seulement pour les types de boîtes auxquels sont associés des tracés :

chaînes de caractères (écrire les caractères),
symboles (tracer le symbole),
fraction (tracer la barre),
racine (tracer le radical),
intégrale (tracer le symbole $\int$),
bloc (tracer les symboles délimiteurs).

Pour construire l'image d'une formule existante, l'éditeur analyse la forme externe de gauche à droite. Chaque fois qu'il rencontre un caractère identifiant un type de boîte, il crée la boîte correspondante avec la taille minimum, comme il le fait sur les commandes de création de l'utilisateur pendant la saisie. Pour chaque boîte créée, il agrandit ou déplace les boîtes englobantes, exactement comme on vient de le voir, la seule différence étant que rien n'est ni effacé de l'écran ni affiché tant que la formule n'est pas complètement traitée. Chaque parenthèse fermante de la forme externe est traitée comme la commande "Fin de construction" de la saisie, c'est à dire en supprimant simplement de la pile la boîte qui est au sommet.

On a vu que pendant les modifications d'une formule préalablement saisie, le réaffichage se faisait, dans une première version, de façon très grossière. Une deuxième version a été développée pour améliorer les temps de réponse de l'éditeur. L'algorithme utilisé lors de la saisie ne pouvait pas être appliqué à ce cas, puisqu'il ignore toutes les boîtes suivant (au sens de la structure) la boîte courante. Nous avons jugé inutile d'augmenter encore la complexité de l'algorithme existant, et un nouvel algorithme, plus simple, a été développé spécifiquement pour les modifications.

Avec ce nouvel algorithme, le réaffichage pendant les modifications se base toujours sur le calcul complet de la nouvelle image à partir de la forme externe, puisque c'est sur la forme externe qu'agissent d'abord toutes les commandes de modification. Mais la nouvelle image n'est pas entièrement affichée à la place de l'ancienne ; seule les différences sont réaffichées. L'ancienne liste de boîtes est sauvegardée avant que la nouvelle ne soit

construite, puis, lorsque la nouvelle liste de boîtes est prête, les boîtes des deux listes sont comparées. L'éditeur parcourt l'ancienne liste et pour chaque boîte, il cherche s'il en existe une identique dans la nouvelle liste. Les boîtes identiques de la nouvelle liste sont marquées comme déjà affichées. Les boîtes de l'ancienne liste qui n'ont pas d'équivalent dans la nouvelle liste sont effacées de l'écran. L'éditeur parcourt ensuite la nouvelle liste et affiche toutes les boîtes qui ne sont pas déjà marquées affichées. L'image sur l'écran est alors à jour et l'ancienne liste n'est plus utile.

6. LES DEVELOPPEMENTS

De façon à valider les concepts qui ont conduit à la réalisation d'Edimath, plusieurs versions de l'éditeur et du formateur ont été développées. L'éditeur du départ a été porté sur plusieurs machines différentes et d'autres applications ont été connectées à l'éditeur.

6.1. Les portages

La première version a été écrite sur LSI-11, en Pascal, sous le système RSX-11M. Dans cette version, l'éditeur utilise un terminal alpha-numérique avec caractères pseudo-graphiques du type VT100. Le formateur utilise une imprimante à aiguilles Sanders qui offre plusieurs polices et des positionnements fins des caractères. La faible puissance de la machine a été un bon stimulant pour l'optimisation de l'algorithme de calcul des boîtes. A partir de cette version de base, deux versions ont évolué en parallèle, l'une pour les écrans à matrice de points, l'autre pour les terminaux alpha-numériques.

Dès 1981, une première opération de portage [Quint 82] a permis d'obtenir, sur le Buroviseur Kayak, une version pour écran graphique. L'intérêt principal de cette version résidait dans la meilleure qualité de l'image présentée sur l'écran et dans un confort accru de l'interface utilisateur, en particulier grâce aux menus. C'est sur cette version qu'ont été expérimentés les changements automatiques de corps des caractères, puisque plusieurs polices pouvaient être utilisées simultanément. En revanche, le manque de puissance de la machine n'a pas permis d'augmenter de façon significative le degré d'interactivité. Ainsi, tout comme dans la version d'origine, l'image présente quelques distorsions temporaires afin d'éviter des calculs trop fréquents : les centrages, par exemple, ne se font pas lors de la frappe de chaque caractère, mais seulement lorsque l'expression à centrer est terminée (en fin de dénominateur pour une fraction, par exemple).

C'est avec la version Perq qu'une amélioration significative a été apportée dans le domaine de l'interactivité. Par rapport au Buroviseur, il a été possible de maintenir un affichage correct de l'image en toutes circonstances. Dans cette version, dès qu'un nouveau

caractère est entré, l'image est immédiatement mise à jour en tenant compte du nouveau caractère. La figure 2.3 montre un exemple de ce que l'utilisateur voit sur l'écran ; c'est une copie agrandie d'une image affichée par la version Perq (les contours des boîtes ne sont évidemment pas visibles sur l'écran).

La dernière évolution de cette version pour écran à matrice de points est celle qui a été adaptée au Macintosh par X. Rousset de Pina [Quint 86d], et qui est dérivée de la version Perq. L'adaptation a essentiellement consisté à prendre en compte l'environnement spécifique du Macintosh. Le processus de sélection a également été modifié pour se rapprocher d'avantage des habitudes des utilsateurs de la machine. La présence d'une souris rend en effet peu naturelle l'extension de la sélection par des touches de fonction.

La deuxième famille des versions d'Edimath est celle qui utilise des terminaux alpha-numériques. La première version de cette famille a été obtenue par portage depuis le LSI-11 vers la SM-90. Elle ne présente aucune différence fonctionnelle par rapport à la version initiale, et elle utilise le même type de terminal. C'est la première version sur un système UNIX.

La même opération a été réalisée sur VAX, d'abord sous VMS, puis sous UNIX. La version UNIX étant la plus utilisée a été développée plus complètement. C'est sur cette version que le deuxième algorithme de réaffichage des modifications a été mis en œuvre. C'est également sur cette version que les terminaux VT220 ont été pris en compte. Cette amélioration constitue un progrès sensible pour l'utilisateur, puisqu'il voit sur l'écran de son terminal une image plus proche de ce qu'il obtiendra à l'impression. Sur un VT100 les caractères grecs ou les symboles mathématiques ne peuvent pas être visualisés : le terminal n'a qu'un jeu de caractères. Les caractères non latins sont donc affichés sous la forme de caractères latins, mais en surbrillance, ce qui n'est pas toujours très parlant. Au contraire, le VT220 peut afficher un deuxième jeu de caractères, chargeable par l'application. Edimath tire parti de cette facilité pour afficher clairement la plupart des lettres grecques et des symboles mathématiques. La version VAX/UNIX comporte également un formateur adapté à l'imprimante à laser LN01. Ce programme a été intégré au formateur Sroff, ce qui permet d'insérer des formules traitées par Edimath dans les documents formatés par Sroff. C'est de cette façon qu'ont été produites les formules qui illustrent ce chapitre.

Entre toutes ces versions, l'image présentée à l'utilisateur varie beaucoup, de même que le comportement de l'éditeur. Mais la représentation externe des formules est strictement la même pour toutes les machines. Ainsi, une formule saisie sur VAX avec un VT100, donc avec un aspect graphique médiocre, peut être éditée ensuite sur un Macintosh ou un Perq avec une bien meilleure qualité graphique, sans autre manipulation que le transfert du fichier contenant la forme externe de la formule. Quelle que soit la machine utilisée pour la saisie, la formule peut être imprimée sur imprimante à laser, avec la même qualité que les formules qui illustrent ce chapitre.

Cette revue des versions d'Edimath ne serait pas complète si on ne mentionnait le système Mathor [Dreyfus] développé par la société Novédit. Il ne s'agit pas d'une version de plus d'Edimath, mais d'un produit différent qui s'inspire directement des résultats de l'expérience Edimath. Les principes de structuration et d'affichage des formules sont les mêmes, mais la réalisation a été faite indépendamment d'Edimath. L'utilisateur retrouve la même souplesse d'utilisation, mais il bénéficie d'un environnement plus large puisque l'éditeur de formules est complètement intégré à un éditeur de texte, ce qui permet de produire directement, avec un seul outil, des documents mêlant harmonieusement des formules et du texte. Edimath, lui, est resté (jusqu'à maintenant) un éditeur de formules uniquement. Les formules qu'il produit peuvent être intégrées dans des documents, notamment dans les versions VAX/Unix et Macintosh, mais ces documents sont manipulés avec d'autres outils.

Si Mathor constitue la première retombée industrielle de la recherche entreprise avec le projet Edimath, d'autres produits se sont également inspirés des résultats du projet. On peut notamment mentionner EasyTEX [Crisanti] et Pléiade [Nanard 86]. Il faut enfin signaler un travail qui s'est déroulé simultannément au projet Edimath et qui a abouti à des résultats comparables [Levison], bien que le prototype décrit soit limité à deux constructions et que l'impression des formules ne soit pas réalisée.

6.2. La connexion à d'autres applications

Edimath a également été utilisé dans d'autres applications que le traitement de documents. Une expérience intéressante, InterMath [Marchand], a notamment été menée pour tirer parti de l'interactivité d'Edimath dans le but d'améliorer le dialogue des utilisateurs avec les systèmes de calcul numérique ou formel.

L'idée consiste à utiliser Edimath pour saisir des formules qui subissent ensuite des traitements mathématiques (calcul, évaluation, dérivation, intégration...). Pour ce genre de traitement, il est clair que la structure retenue par Edimath est insuffisamment précise. Pour cette raison, une fois la formule saisie, une analyse plus fine de la forme externe a lieu, pour en extraire la structure mathématique complète. On recherche ainsi les noms des fonctions, on isole les opérateurs dans les chaînes de caractères, on cherche la variable d'intégration, etc... Cette analyse donne lieu d'une part à la construction d'une structure mathématique riche, sur laquelle se feront les calculs, et d'autre part à une forme externe raffinée, avec notamment un parenthésage complet. Cette nouvelle forme externe est à nouveau passée à Edimath qui l'affiche, permettant à l'utilisateur de constater l'interprétation qui a été faite et de la modifier si une ambiguïté a été mal traduite.

Il s'agit essentiellement d'un système de réécriture d'une grammaire dans une autre, la grammaire de départ étant celle de la forme externe d'Edimath, et la grammaire d'arrivée étant la grammaire d'expressions mathématiques qui a été définie par les auteurs d'InterMath ; c'est une grammaire qui permet de représenter la structure mathématique complète des formules, donc d'en exprimer toute la sémantique et ainsi de les rendre calculables. L'ensemble permet d'offrir à l'utilisateur d'un système de calcul une interface agréable et souple, alors que ces systèmes imposent en général des langages complexes et rigides pour la description des formules.

Une autre expérience d'intégration d'Edimath a été faite pour combiner dans un même éditeur l'édition structurée de formules et de graphique [Joloboff 84]. Le système, réalisé sur Perq, permet d'éditer des schémas structurés en y insérant des formules et du texte, les formules étant manipulées par Edimath.

6.3. La connexion à TEX

Une application d'Edimath a été développée à l'IRISA, par J. André et Y. Gründt [André 85b]. L'idée est d'utiliser Edimath comme un préprocesseur de TEX. On peut en effet considérer que TEX produit des documents d'une grande qualité typographique, et que notamment les formules mathématiques sont particulièrement soignées. Mais, si les résultats produits par TEX constituent une référence en matière de typographie mathématique, le langage d'entrée n'est pas facile à maîtriser et les erreurs sont fréquentes.

Dans cette expérience, on tire également parti de l'interface utilisateur d'Edimath pour saisir et éditer les formules de façon interactive, et la forme externe ainsi produite est ensuite traduite dans le langage d'entrée de TEX. Au contraire de ce qui se passe avec InterMath, qui enrichit les structures des formules, il n'y a pas là de problèmes importants d'ambiguïté au cours de la traduction, puisque les structures source et cible sont de même niveau.

7. CONCLUSION

7.1. Les qualités et les défauts d'Edimath

L'expérience Edimath a clairement montré les avantages offerts par la prise en compte de la structure logique dans l'édition interactive des expressions mathématiques. Cette approche permet notamment

- des traitements évolués sur les formules : formatage mais aussi calcul,
- une interface simple et "naturelle" pour l'utilisateur,
- une réelle indépendance vis à vis du matériel utilisé, aussi bien pour l'édition que pour l'impression.

L'exemple ci-dessous illustre ce dernier point. Il montre la même formule, telle que l'utilisateur la voit sur l'écran d'un terminal alpha-numérique, et, en dessous, telle qu'elle est imprimée sur une imprimante à laser.

$$\frac{1}{2n}\int_{0}^{\sqrt{y}}\left[\sum_{k=1}^{n} \sin^2 x(t)_k\right] f(t)\ dt$$

$$\frac{1}{2n}\int_{0}^{\sqrt{y}}\left(\sum_{k=1}^{n} \sin^2 x_k(t)\right) f(t)\ dt$$

Un autre avantage important qui découle de la représentation structurée des formules est l'ouverture de l'éditeur vers d'autres applications. Une représentation purement graphique, à cause de sa pauvreté sémantique, n'autorise pas d'autre traitement que le simple affichage des formules ou leur impression à l'identique. Cela est un avantage certain sur les systèmes interactifs de traitement de texte qui offrent une extension "mathématique". Ces systèmes sont très fermés dans le sens où un document produit par eux ne peut pas être utilisé dans un autre environnement, et ne peut même pas être imprimé ni édité sur une machine différente de celle qui a servi à produire la version initiale.

Par rapport aux formateurs, Edimath a pour lui l'avantage de l'interactivité : l'utilisateur voit ce qu'il fait. On pourrait lui reprocher une moins bonne qualité de l'image produite. Sur certains écrans, effectivement, l'image n'est pas parfaite. Mais l'écran n'est pas censé montrer l'image finale qui sera obtenue sur le papier. Il sert seulement à contrôler la construction de la formule. Sur une imprimante à laser, on peut obtenir une très bonne qualité d'impression, qui se compare avantageusement à ce que produisent Scribe ou Eqn. De plus, il est possible d'utiliser TEX pour imprimer les formules d'Edimath, et ainsi d'obtenir ce qui est couramment considéré comme la meilleure qualité typographique, au moins pour les mathématiques.

Ces qualités ont d'ailleurs intéressé de nombreux utilisateurs potentiels et des industriels du traitement de texte. Rappelons seulement le développement de Mathor, et l'éditeur Edimath sur Macintosh, diffusé par Microsphère.

Mais, bien qu'il représente un progrès important par rapport aux autres outils d'écriture des mathématiques, Edimath a quelques défauts. Comme tout éditeur structuré, il hérite de la structure qu'il traite une certaine rigidité. Seules les constructions mathématiques qui sont conformes aux constructions d'Edimath peuvent être exprimées aisément. D'autres sont plus difficiles à entrer. Les tenseurs ou les matrices, par exemple, ne peuvent pas toujours être représentés correctement. En fait l'utilisateur peut parfois souhaiter étendre le modèle d'Edimath, en lui ajoutant de nouvelles constructions, ce qui est impossible : ce type d'extension demande des modifications dans le code même de l'éditeur et du formateur. Notons en revanche que, si le matériel le permet, l'utilisateur peut créer ses propres caractères et symboles, dans le cas où les polices utilisées habituellement avec Edimath ne sont pas suffisantes.

La présentation réalisée automatiquement à partir de la structure heurte certains utilisateurs qui souhaitent contrôler tous les détails de l'aspect graphique des formules. Ils voudraient par exemple que, dans certains cas, les bornes d'une intégrale soient affichées non pas sur la droite du symbole $\int$, mais au-dessus et au-dessous. D'autres souhaitent écrire dans un corps plus faible les fractions qui apparaissent à l'intérieur de fractions, bien que ce ne soit pas la règle générale. Edimath n'offre pas ce type de souplesse. On peut toutefois envisager que le mécanisme des attributs, utilisé notamment pour contrôler le mode d'alignement des vecteurs, soit étendu pour traiter ces deux cas. Mais là encore l'extension n'est pas à la portée des utilisateurs puisqu'elle nécessite une modification du code de l'éditeur.

Enfin, le dernier défaut notable d'Edimath, bien qu'il ne soit pas inhérent aux principes de l'éditeur, est le couplage relativement lâche qu'il présente avec les environnements de traitements de documents. Pour offrir un plus grand confort d'utilisation, il faudrait intégrer Edimath dans un système de production de documents, comme cela a été fait dans Mathor.

7.2. Pour un système général

Pour remédier à ces problèmes, et surtout pour élargir le champ d'application de l'édition structurée des documents, nous avons préféré définir un nouveau projet plutôt que d'étendre Edimath. Les concepts mis en œuvre dans l'éditeur de formules avaient prouvé leur intérêt, au moins pour la manipulation des expressions mathématiques ; il s'agissait de les élargir et de les valider dans un contexte de traitement de documents complets.

L'objectif premier du projet Grif était donc de développer un système interactif pour l'édition structurée de documents de tous types, depuis les simples lettres jusqu'aux

volumineux rapports techniques, en passant par les formulaires de toutes sortes et les articles scientifiques.

Pour chacun de ces types de documents le système doit prendre en compte une structure logique adaptée. De même que la structure logique manipulée par Edimath a été conçue pour représenter des expressions mathématiques, de même la structure logique traitée par Grif doit correspondre à l'organisation des différents documents. Pour une lettre elle doit faire intervenir une date, une référence, une adresse de destination, des paragraphes, une signature. Pour un rapport, elle doit prévoir des chapitres, des sections comportant des sous-sections sur plusieurs niveaux, des annexes, des notes de bas de page, des renvois, une table des matières, un index, etc... Selon le document, les types d'éléments nécessaires varient, mais aussi leur organisation : alors qu'une lettre est une simple séquence d'éléments, un rapport peut être vu à travers une structure arborescente dont certains éléments (comme les renvois) ont de plus des relations non-hiérarchiques. Il faut donc traiter une grande variété de types d'éléments (et non plus seulement les huits constructions d'Edimath) et des structures variées (pas uniquement arborescentes). Le système doit donc être ouvert et facilement extensible pour s'adapter à de nombreux types de documents.

Comme dans Edimath, il est souhaitable que l'éditeur connaisse la structure logique que doit respecter le document traité, de façon qu'il puisse guider l'utilisateur en créant pour lui les éléments constitutifs du document. Ce modèle de document doit être souple pour s'adapter à tous les types de documents et à une structure moins rigoureuse que celle des formules : l'organisation des documents n'est pas aussi précise que celle des expressions mathématiques. Cependant cette souplesse doit se combiner à une certaine rigidité dans les cas où il est nécessaire d'imposer la présence de composants indispensables. La structure étant plus variée et moins rigide que pour les formules, il faut que l'utilisateur soit guidé et sache à tout instant les éléments qu'il peut créer, détruire ou déplacer dans le document qu'il traite.

Avec les documents, la structure logique présente certains intérêts qu'elle n'a pas dans les formules. Elle doit notamment permettre des traitements puissants comme les numérotations automatiques de toutes sortes d'éléments (chapitres, sections, formules, théorèmes, exercices) ou encore la constitution d'index ou de tables (des matières, des figures...)

L'aspect graphique des documents diffère aussi de celui des formules, mais le principe adopté pour Edimath, qui construit une image en appliquant des règles de présentation sur une structure logique, doit être repris pour les documents. On peut en attendre les mêmes bénéfices : l'utilisateur est déchargé du travail de présentation et peut se consacrer complètement à la rédaction et à l'organisation de son texte, sans être perturbé par des tâches annexes. Cependant certaines limitations d'Edimath, peu gênantes pour les

formules, doivent être levées dans un système traitant des documents. Il est notamment souhaitable d'obtenir des mises en pages différentes du même document avec le moins possible d'interventions manuelles : l'utilisateur doit avoir le choix des règles de présentation à appliquer au document qu'il traite.

A la différence des formules, les documents ne présentent pas une structure homogène. Ils peuvent inclure des éléments de natures variées, comme des figures, des tableaux, des formules ou des programmes. Un système de traitement de documents doit permettre l'intégration de tels éléments et leur manipulation d'une façon aussi homogène que possible. On doit proposer une autre technique que celle pratiquée avec Edimath, qui consiste à utiliser un éditeur pour les formules et un autre, différent, pour le reste du texte. Au contraire, l'utilisateur doit se trouver dans le même environnement lorsqu'il permute deux colonnes d'un tableau ou deux sections, ou encore lorsqu'il édite une formule ou le texte d'un paragraphe.

S'il est intéressant de traiter le document d'une manière structurée, il est aussi intéressant de traiter de la même façon les éléments de différentes natures qui sont inclus dans le document. Edimath a montré l'avantage d'un tel traitement pour les expressions mathématiques, mais le même avantage peut être obtenu pour les tableaux, les programmes ou certains types de graphiques. Le système doit donc autoriser une manipulation structurée de ces objets, et prévoir que les différentes structures se mêlent : une formule peut apparaître aussi bien dans un paragraphe que dans une cellule d'un tableau ou dans un élément d'un graphique et on doit pouvoir l'éditer de la même façon, où qu'elle se trouve. Pour ce qui concerne les formules, le système doit offrir au moins les mêmes services qu'Edimath.

Il est clair que les dimensions d'un tel éditeur généralisé ne sont pas les mêmes que celles d'un simple éditeur de formules. Aussi l'architecture du système prend-elle une grande importance. Une des préoccupations principales lors de la conception doit être l'indépendance vis à vis du matériel, et ceci pour deux sortes de raisons. D'abord pour des raisons évidentes de portabilité, mais aussi pour des questions touchant à l'interface homme-machine. En effet, il n'existe pas de système interactif traitant des documents structurés de toutes sortes et intégrant complètement des éléments de différentes natures, traités également de façon structurée et à travers une seule et même interface. Faute de référence, la conception d'une interface utilisateur est délicate. Il faut donc prévoir dès le départ une architecture qui se prête à des changements, même profonds, de l'interface homme-machine. Ainsi l'utilisation d'une version expérimentale peut permettre de raffiner les moyens de dialogue et conduire rapidement à une version largement utilisable.

Les grandes lignes de Grif étant esquissées, il reste à réaliser le système. Les deux chapitres suivants sont consacrés à sa mise en œuvre.

CHAPITRE 3

UNE APPROCHE GLOBALE : DOCUMENTS ET OBJETS STRUCTURES

1. PRESENTATION

L'objectif du projet Grif était de construire un système complet d'édition de documents structurés. Dans un premier temps, nous avons élaboré un modèle de document fondé sur les mêmes principes que le modèle des formules d'Edimath : essentiellement une structure logique à partir de laquelle on peut produire, grâce à des règles de présentation, une représentation graphique. Par rapport à Edimath, la principale difficulté vient de ce que les documents ne sont pas aussi rigoureusement structurés que les formules mathématiques et qu'ils sont beaucoup plus variés. C'est ce qui nous a conduit à définir deux langages qui permettent de spécifier respectivement des structures de documents et les présentations de ces documents. On présente dans ce chapitre le modèle de document défini pour Grif, ainsi que les langages de description de structure et de présentation.

La deuxième étape du projet a consisté à réaliser un système interactif mettant en œuvre le modèle et les langages. Le chapitre 4 présente la conception et la réalisation de ce système.

2. UN MODELE SIMPLE

La difficulté commune des travaux de programmation est de définir le niveau d'abstraction convenable pour les objets manipulés. Si ce niveau d'abstraction est trop bas, les traitements sont limités ; s'il est trop élevé, l'utilisateur peut avoir des difficultés à l'assimiler. Les documents n'échappent pas à cette règle générale ; on va le voir en examinant les systèmes de manipulation des documents couramment utilisés.

Les programmes les plus simples traitant des documents sont sans doute les éditeurs et les systèmes de traitement de texte. Dans la plupart d'entre eux, le document est constitué d'une suite de caractères, qui sont aussi bien des caractères de contrôle que des caractères imprimables. Le caractère de contrôle le plus fréquent est celui qui marque la fin de ligne. Notons que ce caractère varie d'un système à l'autre : ce peut être soit le "Line Feed", comme dans UNIX, soit le retour chariot, soit encore la combinaison des deux.

Cette marque permet déjà une certaine structuration du document, mais avec des ambiguïtés, puisque la même marque est généralement considérée par les éditeurs comme un changement de ligne et par les systèmes de traitement de texte comme un alinéa.

D'autres caractères de contrôle permettent de délimiter les pages ("Form Feed") ou de contrôler l'espace horizontal entre les caractères (espace, "Tabulation", "Back Space"). On trouve aussi des caractères non-standard pour l'espacement vertical, notamment pour indiquer des demi-interlignes vers le haut ou vers le bas, ce qui permet de représenter des exposants ou des indices, par exemple.

Ce type de représentation décrit l'aspect physique d'un document, avec son découpage en pages et en lignes et avec tous les espacements horizontaux et verticaux. Il faut noter que le texte du document (les caractères imprimables) et les informations de structuration physique (les caractères de contrôle) sont intimement mêlés.

La raison principale qui a mené à ce type de modèle est la simplicité avec laquelle un document peut être traité par des outils informatiques. Un éditeur peut afficher très simplement le document sur un écran, la plupart des caractères de contrôle étant directement interprétés par le terminal. De même, ces documents peuvent être imprimés sans difficulté. Mais la contrepartie de cette simplicité de traitement est que les opérations possibles sont très limitées.

Avec ce modèle de document, on ne peut pratiquement effectuer que des traitements de chaînes de caractères, en exploitant autant que possible les caractères de contrôle. Dans bien des cas, ces traitements ne correspondent pas aux attentes de l'utilisateur. Par exemple, un programmeur travaillant en Pascal peut avoir besoin de retrouver dans un programme toutes les affectations d'une variable. Avec un éditeur utilisant ce modèle, pour trouver les affectations de la variable I, il devra rechercher la chaîne 'I:=', mais aussi la chaîne 'I :=', puisque Pascal autorise un espace entre le nom de la variable et le signe d'affectation. Cet espace peut être formé d'un seul ou de plusieurs blancs, ou d'une tabulation, ou encore d'un ensemble de tabulations et de blancs ; il peut même contenir des commentaires. Bref, il n'y a pas de moyen sûr de trouver toutes les affectations de I.

Comme le modèle est évidemment très restrictif, de nombreux systèmes tentent de l'enrichir, sans toutefois en changer les bases, ce qui donne des résultats souvent hasardeux. Ainsi, on ne considère plus seulement des entités purement physiques, comme les pages, les lignes, les caractères imprimables et les espaces, mais on ajoute des entités d'un niveau d'abstraction plus élevé, comme des mots, des phrases ou encore des paragraphes. Ces entités cependant restent définies par des caractères imprimables et des caractères de contrôle de présentation.

Prenons l'exemple des mots. Pour le système, un mot est toute suite de caractères imprimables délimitée par des espaces ou des caractères de contrôle. Définition simple mais incomplète, qui ne correspond pas à la notion de mot de l'utilisateur, puisque, par exemple, '2ème' est considéré comme une suite de deux mots séparés par le caractère de contrôle exposant. Certains éditeurs considèrent aussi tous les caractères non alphabétiques comme des séparateurs de mot, mais cela ne fonctionne correctement que dans la langue anglaise. Les autres langues utilisent habituellement des caractères non-alphabétiques pour représenter leurs caractères spécifiques. L'exemple le plus courant est sans doute celui des lettres 'é' et 'è' en français, pour lesquelles on utilise souvent les codes des accolades ; ainsi les 'e' accentués du français ne sont-ils pas traités comme des caractères alphabétiques et un mot français qui contient des 'e' accentués est vu par un éditeur comme plusieurs mots.

La notion de phrase est également basée sur le même principe des caractères délimiteurs : la fin d'une phrase est marquée par un symbole de ponctuation (point, point d'exclamation, point d'interrogation) suivi d'un espace. Làaussi la définition est incomplète, puisque des initiales risquent d'être interprétées comme des fins de phrase. Le paragraphe est aussi une notion difficile à interpréter en termes de caractères de contrôle de présentation physique : une ligne vide ou une ligne commençant par un espace n'est pas nécessairement la marque d'un début de paragraphe dans l'esprit de son auteur.

Si les éditeurs et les systèmes classiques de traitement de texte tentent d'étendre, même maladroitement, leur modèle de représentation des documents, c'est précisément que le modèle de base est insuffisant pour traiter efficacement un document. Ainsi il serait très fastidieux au cours de l'édition de se déplacer dans un document seulement par pages, lignes et caractères, alors que des mouvements de mot en mot, de phrase en phrase ou de paragraphe en paragraphe sont plus naturels pour l'utilisateur. De même il est plus confortable de pouvoir effacer ou copier un mot, une phrase ou un paragraphe, qu'une page, une ligne ou un caractère.

En fait ce modèle simple est essentiellement orienté vers la représentation de documents "papier" : il décrit l'aspect que doit avoir le document une fois imprimé ou affiché sur un écran. Il ne s'agit pas encore de documents "électroniques", c'est à dire susceptibles d'être transmis entre équipements différents et prêts à être traités par des applications variées.

3. UN MODELE DE HAUT NIVEAU

De toutes les remarques précédentes se dégage une conclusion : le modèle représentant le document comme une suite de caractères imprimables auxquels se mêlent des caractères de contrôle de présentation physique est trop limité pour autoriser des traitements puissants. Un modèle d'un plus haut niveau d'abstraction est nécessaire.

3.1. Un modèle logique

Un modèle de haut niveau doit permettre à l'utilisateur d'agir sur les entités qu'il a à l'esprit lorsqu'il travaille sur un document, même si les traitements réalisés par les outils informatiques sont rendus plus complexes. Le modèle doit donc prendre en compte ces entités en tant que telles. Il s'agit essentiellement d'entités logiques, comme des paragraphes, des sections, des chapitres, des notes, des titres, des références et toutes les entités qui permettent de structurer logiquement un document.

Lorsque l'auteur écrit, c'est ce modèle qu'il a en tête. Il choisit de découper son document en chapitres, et donne un titre à chacun. Le contenu des chapitres est à son tour découpé en sections, sur plusieurs niveaux. Le texte est organisé en paragraphes successifs, en fonction du contenu. Dans cette phase de création, les notions de ligne ou de page, les marges et autres espaces n'ont pas grande importance. Au contraire, si le système impose de décrire les documents en ces termes, cela constitue plutôt une gêne.

Pour construire un modèle de ce niveau, il faut essentiellement définir :

- les entités mises en jeu dans les documents,
- les relations qu'elles ont entre elles.

Un document est souvent créé de façon non linéaire, avec des retours en arrière, des hésitations, des changements de structure, des déplacements de texte. Ces aspects dynamiques doivent être pris en compte par un système de traitement de documents. Le modèle de document, lui, doit seulement permettre d'exprimer les entités sur lesquelles le système agit.

Le choix des entités à intégrer dans le modèle est délicat. Pour tel document on a besoin de chapitres, pour tel autre on utilise seulement des sections de différents niveaux. Certains documents contiennent des annexes, d'autres pas. Selon les documents, la même entité peut porter des noms différents comme "Introduction" ou "Présentation". Certaines entités, indispensables dans certains cas, comme les clauses dans un contrat, ou l'adresse du destinataire dans une lettre, sont sans utilité dans la plupart des autres cas.

Les différences entre documents ne résident pas seulement dans les entités mises en jeu, mais aussi dans leurs relations, dans la façon dont elles sont organisées entre elles. Dans certains documents, les notes se trouvent dispersées au fil du document, par exemple au bas de la page où se trouve leur renvoi, dans d'autres documents elle sont regroupées à la fin de chaque chapitre, ou encore en fin d'ouvrage. Pour certains documents, l'introduction peut être composée de plusieurs sections, pour d'autres, elle est limitée à une courte suite de paragraphes.

Tout cela rend improbable la définition d'un modèle unique pouvant décrire n'importe quel document à un niveau d'abstraction relativement élevé. On pourrait évidemment tenter de dresser une liste d'entités d'usage général, comme les chapitres, sections, paragraphes, titres, etc..., et ramener les entités exclues de cette liste à celles qui y figurent. Ainsi une introduction pourrait être assimilée à un chapitre, une clause de contrat à un paragraphe ou une section. Mais en voulant élargir le domaine d'utilisation de certaines entités privilégiées, on risque de les vider de leur sens et donc de réduire la puissance du modèle. Et puis, cela résoudrait partiellement le problème du choix des entités, mais pas celui de leur organisation : si un chapitre doit clairement comporter des sections, comment indiquer qu'une introduction n'en comporte pas, puisqu'on l'assimile à un chapitre ? Une solution consisterait à compter l'introduction dans la liste des entités de base. Mais alors comment distinguer, selon les documents, les introductions qui comportent des sections de celles qui n'en comportent pas ? Peut-être en définissant deux types d'introduction... Dans ce cas, la liste des entités d'usage général risque de s'étendre indéfiniment.

3.2. Un modèle ouvert

Il paraît impossible de dresser un inventaire exhaustif de toutes les entités nécessaires et suffisantes pour décrire n'importe quel document. Il semble tout aussi impossible de décrire une fois pour toutes l'organisation de ces entités dans un document. C'est pourquoi, l'approche que nous avons prise s'écarte de celle utilisée dans certains systèmes de traitement de documents structurés, comme Mentor-Rapport [Mélèse] ou COBATEF [Pœls], où les entités et leur organisation sont spécifiées de façon définitive. Nous proposons plutôt un *méta-modèle* qui permet de décrire de nombreux *modèles*, chaque modèle ayant une utilisation limitée à une *classe* de documents.

On appelle *classe* un ensemble de documents ayant des structures très proches. Ainsi, l'ensemble des rapports de recherche publiés par un laboratoire constitue une classe ; l'ensemble des propositions commerciales établies par le service des ventes d'une société constitue une autre classe ; l'ensemble des articles publiés par une revue scientifique donnée constitue une troisième classe. Il est clair qu'on ne peut pas dresser un inventaire de toutes

les classes de documents possibles et qu'on peut être amené à en créer de nouvelles selon les besoins.

Pour donner une définition plus rigoureuse des classes, il faut introduire les notions de *structure générique* et de *structure spécifique*. Chaque document a une *structure spécifique* qui organise les différentes parties dont il est constitué. Illustrons cela à l'aide d'un exemple simplifié (voir figure 3.1). Considérons deux rapports, A et B. Le rapport A comprend une introduction suivie de trois chapitres et d'une conclusion. Le premier chapitre comprend deux sections, le deuxième trois sections, etc... Cela constitue la structure *spécifique* du document A. On peut présenter de même la structure spécifique du document B : une introduction, deux chapitres, une conclusion. Le chapitre 1 est composé de trois sections et le chapitre 2 de quatre. Les structures spécifiques de ces deux documents sont donc différentes.

Rapport A
- Introduction
- Chapitre 1
  - Section 1.1
  - Section 1.2
- Chapitre 2
  - Section 2.1
  - Section 2.2
  - Section 2.3
- Chapitre 3
- Conclusion

Rapport B
- Introduction
- Chapitre 1
  - Section 1.1
  - Section 1.2
  - Section 1.3
- Chapitre 2
  - Section 2.1
  - Section 2.2
  - Section 2.3
  - Section 2.4
- Conclusion

Figure 3.1 : Deux structures spécifiques.

La *structure générique* définit le mode de construction des structures spécifiques. Les documents A et B, bien que différents, sont construits selon la même structure générique, qui spécifie qu'un rapport comporte une introduction suivie d'un nombre variable de chapitres et d'une conclusion, chaque chapitre comportant un nombre variable de sections.

Il y a une correspondance biunivoque entre une classe et une structure générique : tous les documents d'une classe sont construits d'après la même structure générique. D'où la définition de la classe : une classe est un ensemble de documents dont la structure spécifique est construite d'après la même structure générique. Une classe est caractérisée par sa structure générique.

Ainsi, une structure générique peut être considérée comme un modèle du niveau qui nous intéresse, mais seulement pour une classe de documents. Si on se limite à une classe, il est alors possible de définir un modèle (disons maintenant une structure générique) qui représente bien les documents de la classe, avec les entités nécessaires, et sans s'encombrer d'entités inutiles. La description de l'organisation des documents de la classe est alors suffisamment précise.

3.3. Un modèle séparant les structures logique et physique

On a vu que le principal problème posé par le modèle le plus simple était l'imbrication des niveaux physique et logique dans une seule représentation du document. Avec les structures génériques, on ne considère que l'organisation logique des documents, pas leur présentation physique sur un écran ou des feuilles de papier. Le problème de l'imbrication des deux niveaux ne se pose donc plus. Cependant, il faut bien prendre en compte la présentation des documents dès qu'il s'agit de les afficher ou de les imprimer.

Si on considère les documents imprimés courants, on constate que les détails de présentation servent essentiellement à faire ressortir leur structure. Hormis quelques domaines particuliers, comme la publicité notamment, la présentation est rarement indépendante de l'organisation logique du texte. Tout l'art du typographe consiste d'ailleurs à mettre en relief l'organisation du texte qu'il compose, sans pour autant accrocher l'œil du lecteur avec des effets trop marqués. Ainsi on utilise l'italique ou le gras pour mettre en valeur certains mots ou expressions qui ont une signification plus forte que le reste du texte : mots-clés, nouvelles notions, citations, titres d'ouvrages, etc... D'autres effets concernent l'organisation du texte : espacement vertical, changement de marge, saut de page, centrage, éventuellement combinés à des changements de corps et/ou de graisse des caractères. Ces effets servent à indiquer les changements de paragraphe, de section ou de chapitre : le niveau dans la structure du document est d'autant plus élevé que l'effet est plus marqué.

Puisque le modèle permet de décrire toute la structure logique du document, la présentation peut se faire à partir du modèle, sans être immergée dans le document lui-même. Il suffit d'utiliser la structure du document pour effectuer les changements voulus dans sa présentation : changement de corps, de graisse, espacements, marges, centrages, etc...

De même qu'on ne peut pas définir une structure générique unique pour tous les documents, de même on ne peut pas définir des règles de présentation universelles qui s'appliqueraient à toutes les classes de documents : pour tel type de document les titres de chapitres seront centrés dans la page et imprimés en gras dans un corps important, pour tel

autre document ces mêmes titres de chapitres seront imprimés en italique dans un corps plus faible, et alignés sur la marge gauche.

Il faut donc spécifier la présentation des documents selon leur classe, ce qui peut être fait de façon fine, puisque la présentation peut s'exprimer en fonction des entités définies dans la structure générique de la classe. Ainsi il est possible de préciser une présentation différente pour les titres de chapitre et les titres de section, et même pour les différents titres de section selon leur niveau hiérarchique. L'ensemble des règles qui spécifient la présentation de tous les types d'éléments définis dans une structure générique est appelé une *présentation générique*.

Les bénéfices tirés d'une présentation liée à la structure générique et décrite par une présentation générique sont multiples. Il y a d'abord l'homogénéité. Puisque tous les documents d'une classe répondent à la même structure générique, une présentation homogène des différents documents de la même classe peut être assurée en appliquant la même présentation générique à tous les documents de la classe. L'homogénéité de présentation s'applique également à l'intérieur d'un document : tous les titres de section du document seront présentés de la même façon, la première ligne de tous les paragraphes aura le même retrait, etc...

Un autre avantage de cette approche de la présentation est qu'elle facilite les changements de l'aspect graphique des documents. Il suffit de modifier, dans la présentation générique, les règles attachées à chaque type d'entité pour transformer la présentation de tout un document, et de façon homogène pour ce document. Dans ce cas, évidemment, l'homogénéité à l'intérieur de la classe n'est plus assurée, mais le moyen de la contrôler est simple, c'est l'unicité de la présentation générique pour toute la classe.

Si l'homogénéité de présentation de la classe n'est pas requise de façon stricte, on peut ainsi adapter l'aspect du document selon l'utilisation qui en est faite ou selon l'appareil utilisé pour le traiter. Il suffit pour cela de disposer de plusieurs présentations génériques différentes pour une classe de documents. En appliquant l'une ou l'autre de ces présentations au document, on peut le voir sous différents aspects graphiques. Il faut souligner que ce type de modification de la présentation n'est pas un changement dans le document lui-même (dans sa structure logique ou son contenu), mais seulement dans son apparence au moment de l'édition ou de l'impression.

3.4. Un modèle général

Nous n'avons jusqu'ici abordé que la structure globale des documents, sans envisager le contenu de cette structure. On pourrait se limiter à un contenu purement textuel en considérant qu'un titre ou un paragraphe contient un simple texte linéaire. Mais ce modèle serait trop restrictif. En effet, certains documents ne contiennent pas uniquement du texte, mais également des tableaux, des schémas, des photographies, des formules mathématiques, des fragments de programme. Le modèle doit donc être élargi pour permettre la représentation de tels *objets*.

De la même façon que pour l'ensemble d'un document, on essaie de prendre en compte la structure logique des objets de ce genre. Certains sont clairement structurés, d'autres le sont moins. On peut reconnaître une structure dans les formules mathématiques (comme le fait Edimath), dans les tableaux, ou dans certains types de schémas. En revanche, il est plus difficile de définir la structure d'une photographie ou de certains dessins. Mais dans tous les cas, il ne semble pas possible de définir une structure unique qui puisse représenter l'ensemble de ces types d'objets. L'approche qui a mené à la définition d'une méta-structure et de classes pour les documents peut tout à fait être reprise pour les objets. On définit des classes d'objets qui regroupent les objets de même nature, construits à partir de la même structure générique.

Ainsi, on peut définir une classe mathématique (classe d'objets formules) et utiliser la structure manipulée par Edimath comme structure générique de la classe. Mais si, avec une structure générique unique, on peut représenter des formules mathématiques suffisamment variées, pour d'autres objets, structurés de façon moins rigoureuse, il est nécessaire de définir plusieurs classes. Cela assure, comme pour les documents, que le modèle correspond bien aux objets que l'on veut représenter et permet de prendre en compte des objets qui n'auraient pas été prévus initialement. Cette remarque peut d'ailleurs s'appliquer également aux formules, ce qui résoudrait un des problèmes d'Edimath, en adaptant la structure générique considérée aux besoins spécifiques de certains utilisateurs.

On peut envisager plusieurs types de tableaux ou plusieurs types de schémas et, pour chacun de ces types, définir une classe d'objets. Cela conduit, par exemple dans le domaine des schémas, à une classe organigramme, à une classe schéma fonctionnel, à une classe carte syntaxique. Mais le modèle n'est pas trop contraignant et on peut aussi définir une classe qui engloberait ces trois classes-là.

Puisqu'on retient pour les objets le même niveau de représentation logique que pour les documents, on peut en tirer les mêmes avantages. En particulier, il est possible de définir la présentation de façon externe aux objets (instances) et de la relier à la classe. Ainsi, comme pour les documents, on aura une présentation uniforme des objets de même nature et on pourra changer globalement la présentation de tous les objets d'une classe

donnée, simplement en changeant la présentation générique de la classe. Autre avantage, un système utilisant ce modèle de document ne perturbera pas l'utilisateur avec des tâches de présentation, mais au contraire, il le laissera se concentrer sur l'aspect logique du document et des objets.

Il est clair que tous les documents d'une classe n'utilisent pas nécessairement les mêmes classes d'objets : dans un rapport technique on utilisera des tableaux, dans un autre on utilisera des formules mathématiques, mais pas de tableaux. Les classes d'objets utilisables ne doivent pas être mentionnés de manière limitative dans la structure générique des documents ; au contraire, il doit être possible de les choisir librement, indépendamment de la classe du document, parmi un large ensemble.

Si les classes d'objets sont ainsi banalisées et utilisables dans tout document, on peut élargir la notion d'objet et accepter sous ce vocable non seulement des éléments non textuels, mais aussi certaines variétés d'éléments textuels qui peuvent apparaître dans pratiquement tous les documents, quelle que soit leur classe. Parmi ces éléments textuels, on peut mentionner un certain nombre d'éléments définis dans Scribe comme des environnements : énumération, description, exemple, citation, etc... voire même les paragraphes.

On définit donc le modèle de document non pas comme un modèle unique et général décrivant d'un coup toutes les variétés de documents, mais plutôt comme un méta-modèle qui permet de décrire de la même façon plusieurs modèles représentant chacun, soit une classe de documents semblables, soit une classe d'objets similaires que tout document peut intégrer.

4. LA META-STRUCTURE DES DOCUMENTS

S'il apparaît que le concept de méta-structure convient bien à la description des documents au niveau où on l'envisage, il reste à définir précisément cette méta-structure. Cela peut se faire en deux étapes : en choisissant d'abord les éléments à partir desquels on va composer des documents et des objets structurés, en précisant ensuite les moyens d'assembler ces éléments de base dans des structures représentant des documents ou des objets complets.

4.1. Les types de base

Au niveau le plus bas de la structure d'un document, on peut d'abord considérer que l'atome est le caractère. Comme les caractères sont rarement isolés mais font plutôt partie d'une suite linéaire, et pour alléger les structures, on préfèrera retenir comme atome la *chaîne de caractères* et on regroupera dans une même chaîne les caractères consécutifs appartenant au même élément structurel.

Si on ne raffine pas la structure d'un document jusqu'à descendre au niveau des mots ou des phrases, on pourra considérer que le contenu d'un simple paragraphe est une unique chaîne de caractères. En revanche, le titre d'un chapitre, le titre de la première section de ce chapitre et le texte du premier paragraphe de cette section constitueront trois chaînes de caractères différentes, puisqu'appartenant à des éléments structurels distincts. Si au contraire on cherche une représentation très fine de la structure d'un document, on pourra définir des chaînes ne contenant qu'un mot, voire même un seul caractère. C'est le cas par exemple des programmes, pour lesquels on voudrait retenir une structure très proche de la syntaxe du langage de programmation. Dans ce cas, une instruction Pascal d'affectation initialisant une variable simple à zéro comportera deux éléments structurels, l'identificateur de la variable (une courte chaîne de caractères) et la valeur affectée (une chaîne d'un seul caractère, '0').

La chaîne de caractères n'est pas le seul atome nécessaire pour représenter les documents qui nous intéressent. Elle suffirait pour les documents purement textuels, mais dès que les objets que nous avons considérés interviennent, il faut d'autres atomes ; et selon les classes d'objets que l'on veut représenter, une variété plus ou moins grande d'atomes est nécessaire.

Des *éléments graphiques* de base sont utiles pour les tableaux et les schémas de différents types. Ces éléments sont des formes géométriques simples comme des traits horizontaux ou verticaux, suffisants pour les tableaux, ou encore des lignes obliques, des flèches, des rectangles, des cercles, des ellipses, des polygones, pour les schémas. A partir de ces éléments et des chaînes de caractères, on peut construire les objets graphiques et les tables.

Les photographies, bien que peu structurées, doivent pouvoir être représentées dans les documents. Pour cela il est nécessaire de disposer d'éléments d'*images*, représentés par des matrices de points.

Enfin les notations mathématiques font appel à des éléments qui tiennent à la fois du caractère et de l'élément graphique, qui sont les *symboles*. A titre d'exemple on peut citer les radicaux, ou les signes d'intégration, ou encore les grandes parenthèses.

Finalement, nous retenons comme éléments de base pour la construction de documents et d'objets structurés :

- les chaînes de caractères,
- les éléments graphiques,
- les images,
- les symboles mathématiques.

4.2. Les éléments construits

Un document est évidemment formé d'éléments de base. Mais notre modèle comprend aussi des éléments de plus haut niveau. Ainsi dans un document formé de plusieurs chapitres, chaque chapitre est un élément, et dans les chapitres chaque section est également un élément, et ainsi de suite. Un document est donc un ensemble organisé d'éléments.

Il y a dans un document différentes sortes d'éléments. Chaque élément a un *type* qui indique le rôle de l'élément dans l'ensemble du document. Ainsi, nous aurons par exemple les types chapitre et section. Le document est formé d'éléments typés : des éléments de type chapitre et des éléments de type section, entre autres, mais aussi des éléments de type chaîne de caractères ou de type graphique : les éléments de base sont également des éléments typés. A l'opposé, le document lui-même peut aussi être considéré comme un élément typé.

La différence importante entre les éléments de base et les autres éléments du document est que les éléments de base sont des atomes (ils ne peuvent pas être décomposés), alors que les autres, appelés *éléments construits* sont composés d'autres éléments, qui peuvent être aussi bien des éléments de base que des éléments construits. Un élément construit de type chapitre (on dit plus simplement "un chapitre") est composé de sections, qui sont également des éléments construits. Un paragraphe, élément construit, peut être composé de chaînes de caractères, qui sont des éléments de base, et de formules, qui sont des éléments construits.

On peut ici faire un parallèle entre les types d'éléments et les types de données des langages de programmation typés, comme Pascal. Nos quatre types de base peuvent se comparer aux types de base standard de Pascal : entier, caractère, booléen, réel. Les types d'éléments construits peuvent se comparer aux types structurés de Pascal, que chaque programmeur définit à partir des types de base.

Un document est également un élément construit. Cette remarque a son importance, elle permet notamment de considérer un document comme une partie d'un autre document, et inversement, de traiter une partie d'un document comme un document complet. Ainsi, un article proposé à une revue est traité par son auteur comme un document en soi, alors que l'éditeur de la revue le considère comme une partie d'un fascicule. Un tableau ou une figure appartenant à un document peuvent en être extraits et traités comme un document complet, par exemple pour préparer un transparent pour une conférence. On verra que cette banalisation de la notion de document est importante pour un système de production de documents, par la souplesse qu'elle offre à l'utilisateur.

Ces considérations sur les types et les éléments construits s'appliquent aussi bien aux objets qu'aux documents. Un tableau est un élément construit formé d'autres éléments construits, des lignes et des colonnes. Une ligne ou une colonne est formée de cellules, qui sont également des éléments construits contenant des éléments de base (des chaînes de caractères) et/ou des éléments construits commes des formules.

4.3. Les constructeurs

Après les éléments de base et les éléments construits, il faut définir les types d'organisation qui permettront de construire des structures. On s'appuie pour cela sur la notion de *constructeur*. Un constructeur définit un mode d'assemblage de certains éléments dans une structure. Il se place bien au niveau de la méta-structure : il ne décrit pas les relations existant dans une structure donnée, mais définit de quelle façon on peut assembler des éléments pour construire une structure conforme à un modèle. Par comparaison avec Pascal, nos constructeurs correspondent aux méthodes de structuration des types de données : tableaux, enregistrements, ensembles et fichiers.

Pour l'organisation globale des documents, deux constructeurs paraissent immédiatement indispensables : l'agrégat et la liste.

L'*agrégat* permet de définir la structure d'un type d'élément construit comme étant la collection ordonnée d'un nombre donné d'éléments (construits ou de base), chacun de ces éléments ayant un type déterminé. Un rapport (un élément construit du type rapport) a une structure d'agrégat : il est formé d'un titre, d'un nom d'auteur, d'une introduction, d'un corps et d'une conclusion, c'est à dire de cinq éléments typés. Ce type de construction se retrouve dans pratiquement tous les documents, et en général à plusieurs niveaux dans un document. Il correspond à l'enregistrement (`record`) de Pascal.

La *liste* permet de définir la structure d'un type d'élément construit comme étant la suite ordonnée d'éléments (construits ou de base) ayant tous le même type. Le nombre des éléments d'une liste peut être limité aussi bien inférieurement que supérieurement. Il peut

aussi être totalement libre. Le corps d'un rapport est une liste de chapitres et on peut limiter inférieurement à 2 le nombre d'éléments de cette liste (un chapitre est-il vraiment utile s'il n'y en a qu'un dans le rapport ?) Le chapitre lui-même peut contenir une liste de sections, chaque section contenant une liste de paragraphes. La liste correspond au tableau (`array`) à une dimension de Pascal. Au même titre que l'agrégat, la liste est un constructeur très fréquemment utilisé dans toutes sortes de documents. Mais ces deux constructeurs ne sont pas suffisants pour décrire les structures des documents ; d'autres viennent donc les compléter.

Le constructeur *choix* permet de définir la structure d'un type d'élément comme étant l'une parmi plusieurs possibles. Ainsi, un paragraphe peut être soit un simple paragraphe de texte, soit une énumération, soit une citation. Le choix peut se comparer aux parties variantes des enregistrements Pascal (`case` dans un `record`).

Le constructeur choix indique la liste complète des structures possibles, ce qui peut être trop restrictif dans certains cas, comme dans celui du paragraphe pris comme exemple. C'est pourquoi, nous avons introduit le constructeur *unité* qui permet de choisir le type d'un élément plus librement. Si un paragraphe est formé d'unités, il est alors possible de mettre à la place d'un paragraphe un tableau, une formule ou n'importe quel autre objet. En utilisant soit le choix, soit l'unité, on peut définir une structure générique relativement contraignante, ou au contraire très ouverte.

L'unité n'a pas d'équivalent en Pascal. Elle représente un élément dont le type peut être

- soit un type de base,
- soit un objet défini par une structure générique quelconque,
- soit un élément défini dans la structure générique du document comme unité exportable vers les objets inclus dans le document.

Ainsi, par exemple, si un renvoi à une note est défini dans la structure générique Article comme une unité exportable, un tableau (objet appartenant à une autre structure générique) peut comporter des renvois à des notes, lorsqu'il figure dans un article. Utilisé dans un autre type de document, un tableau défini par la même structure générique peut comporter d'autres types d'éléments, spécifiques au type du document où il s'insère. Il suffit pour cela de déclarer, dans la structure générique des tableaux, que le contenu des cellules est formé d'unités. De cette façon, la structure générique des objets est rendue réellement partageable entre différents types de documents.

La *référence* est le dernier constructeur. Il permet de définir un élément d'un document comme étant un renvoi à une autre partie du document, comme le renvoi à une section, à un chapitre, à une citation bibliographique, ou à une figure. La référence peut

être comparée au pointeur de Pascal, mais, à la différence du pointeur, la référence est bidirectionnelle : par exemple un renvoi fait référence à un chapitre et ce chapitre est référencé par ce renvoi.

Pour construire un document on utilise donc quatre constructeurs :

- l'agrégat,
- la liste,
- le choix, et une extension, l'unité,
- la référence.

Pour les objets également ces quatre constructeurs sont suffisants. Il permettent de décrire la structure des objets que nous considérons. Ainsi, avec ces constructeurs, on obtient un méta-modèle homogène qui peut décrire à la fois l'organisation globale d'un document et celle des objets de différentes natures qu'il contient. On verra, après la présentation du langage de description des structures logiques, quelques exemples qui mettent en évidence cette propriété du modèle.

Les trois premiers constructeurs (agrégat, liste et choix) conduisent à des structures arborescentes pour les documents et les objets, les objets étant simplement des sous-arbres de l'arbre d'un document (ou même d'autres sous-arbres d'objets). Le constructeur référence, lui, introduit d'autres relations qui viennent s'ajouter à celles de l'arbre : quand un paragraphe fait référence à un chapitre ou à une section, cette relation sort de la structure purement arborescente. Or les arbres présentent des propriétés intéressantes et les outils pour les manipuler sont bien connus. En revanche, les graphes que constituent les références sont d'un traitement moins facile. On essaie donc de traiter de façon particulière ces relations de référence, pour éviter de considérer les structures de documents comme des graphes tout à fait généraux. On traite simplement des arbres dont certains éléments peuvent avoir d'autres relations, non hiérarchiques, introduites par les références.

4.4. Les éléments associés

Grâce aux trois premiers constructeurs, on spécifie de façon stricte l'organisation du document, avec des éléments construits et des éléments de base. Mais un document ne comporte pas que des éléments aussi clairement reliés entre eux. Il existe des éléments dont la position par rapport à la structure du document n'est pas déterminée. C'est notamment le cas des figures ou des notes. Une figure peut être référencée en plusieurs points du même document et sa place dans le document physique peut varier au cours de la vie du document sans rien changer au sens ou même à la clarté du document. Elle peut être placée une fois

en fin de document, avec toutes les autres figures, une autre fois en haut de la page qui suit la première référence. Les figures peuvent être dispersées à travers le document ou regroupées ensemble. Il en est de même pour les notes, qui peuvent être imprimées en bas de la page où elles sont référencées ou regoupées en fin de chapitre ou encore en fin d'ouvrage. Bien sûr, il s'agit là de la position physique des éléments dans les documents mis en page, mais cela reflète l'instabilité structurelle de ces éléments. On ne peut pas les traiter de la même façon que des éléments comme les paragraphes ou les sections, dont les positions dans la structure sont directement liées à la sémantique du document.

Ces éléments dont la position n'est pas fixée de façon unique dans la structure du document, mais qui en font toutefois partie, sont appelés les *éléments associés*. Cela n'empêche pas qu'ils soient également structurés, c'est à dire que leur contenu soit organisé logiquement par des constructeurs, à partir d'éléments de base et d'éléments construits.

Il peut arriver que des éléments associés soient totalement déconnectés de la structure du document, comme peut l'être un commentaire ou une appréciation d'un lecteur sur l'ensemble du document. Mais le plus souvent, les éléments associés sont reliés au contenu du document par des références. C'est généralement le cas des notes ou des figures, notamment.

Les éléments associés introduisent donc une nouvelle utilisation pour le constructeur référence. Celui-ci ne sert pas seulement à établir des liens entre éléments de la structure principale du document, il sert aussi à relier les éléments associés à la structure principale.

4.5. Les attributs

Tout l'aspect logique des documents n'est pas entièrement décrit par la structure. Certaines informations d'ordre sémantique, mais non exprimées explicitement dans le texte du document, méritent d'être prises en compte. Ce sont notamment celles qui sont habituellement reflétées par des effets typographiques qui ne correspondent pas à un changement d'élément structurel. En effet, certains titres peuvent être mis en gras ou en italique, ou être imprimés avec un style de caractère différent du reste du texte, pour marquer qu'il s'agit d'un élément structurel distinct. Mais ces mêmes effets apparaissent parfois au milieu d'un texte continu, à l'intérieur d'un paragraphe, par exemple. Dans ce cas, il n'y a pas de changement d'élément de structure ; il s'agit de la mise en évidence d'un mot, d'une expression ou d'une phrase. C'est pour exprimer ce type d'information que nous utilisons la notion d'*attribut*.

Un attribut est une information attachée à un élément structurel qui s'ajoute au type de l'élément et en précise la fonction dans le document. Les mots-clés, les mots d'une langue étrangère, les titres d'œuvre qui apparaissent dans le fil du texte sont autant de

chaînes de caractères portant un attribut. Mais les attributs ne s'attachent pas seulement aux éléments de base, il peuvent aussi porter sur des éléments construits. Ainsi un attribut indiquant la langue peut être attaché à un simple mot aussi bien qu'à toute une partie d'un document écrit en plusieurs langues.

En fait, on traite comme un attribut une information qui est liée à une partie d'un document et qui peut être utile à toute application qui travaille sur le document. Par exemple, la langue dans laquelle est écrit le document détermine le choix du jeu de caractères pour un éditeur ou un formateur, elle détermine aussi l'algorithme ou le dictionnaire de coupure des mots à utiliser. L'attribut "mot-clé" facilite le travail d'un système documentaire. L'attribut "mot d'index" permet à un formateur de construire automatiquement un index à la fin du document.

Les attributs peuvent être affectés de deux façons différentes aux éléments qui constituent le document. Ils peuvent être mis dynamiquement et librement, par l'auteur du document, sur n'importe quelle partie, pour y attacher un complément d'information de son choix. Ils peuvent également être affectés automatiquement par toute application traitant le document, y compris un éditeur, pour ajouter systématiquement ce même type d'information à certains éléments prédéfinis du document. A titre d'exemple, si dans un rapport comportant un résumé en français et un résumé en anglais, chacun de ces deux résumés est défini comme une suite de paragraphes, avec, pour le premier résumé, la valeur imposée "français" pour l'attribut "langue", et, pour le deuxième, la valeur imposée "anglais" pour ce même attribut.

De même que les types d'éléments construits sont définis dans chaque structure générique, et non pas dans le méta-modèle, de même les attributs et les valeurs qu'ils peuvent prendre sont définis dans chaque structure générique, selon les besoins de la classe de documents ou de la nature d'objet.

4.6. Discussion du modèle

Les notions d'attribut, de constructeur, d'élément structuré, d'élément associé, sont utilisées dans la définition des structures génériques des documents et des objets. Le problème est de les regrouper pour constituer ces structures génériques. En effet de nombreux types d'éléments et de nombreux attributs se retrouvent dans plusieurs structures génériques. Plutôt que de les redéfinir pour chaque structure où ils apparaissent, il convient de les partager entre ces structures. Les classes d'objets remplissent déjà cette fonction de partage. Si on définit une classe mathématique, on peut utiliser des formules dans plusieurs classes de documents, sans en redéfinir la structure pour chaque classe. Ce problème ne se pose pas seulement pour les objets que nous avons considérés jusqu'ici, il se pose aussi pour

des éléments textuels banalisés que l'on retrouve dans de nombreuses classes de documents. C'est la raison pour laquelle la notion d'objet a été élargie, et que des paragraphes ou des énumérations sont également considérés comme des objets. Ces classes d'objets permettent le partage non seulement des structures d'éléments, mais aussi des attributs définis dans les structures génériques.

La structure, telle qu'elle est présentée ici, peut paraître très rigide, et on peut imaginer qu'un système de production de documents fondé sur ce modèle se révèle très contraignant pour l'utilisateur. C'est en effet un reproche qui est souvent adressé aux éditeurs dirigés par la syntaxe. Pour trois raisons principalement, ce défaut peut être évité avec notre modèle :

- les structures génériques ne sont pas figées dans le modèle lui-même,
- le modèle prend en compte la dynamique des documents,
- les constructeurs offrent une grande souplesse.

Puisque la structure générique d'un document n'est pas prédéfinie, mais au contraire conçue spécifiquement pour chaque classe de documents, elle peut être très précisément adaptée aux besoins. En cas d'inadéquation de la structure générique à un document particulier de la classe, il est toujours possible, soit de créer une nouvelle classe avec une structure générique bien adaptée au nouveau cas, soit d'étendre la structure générique de la classe existante pour prendre en compte les spécificités du document qui pose problème. Ces deux solutions s'appliquent également aux objets dont les structures génériques se révéleraient mal adaptées.

Le modèle doit être naturellement souple pour prendre en compte toutes les étapes de la vie du document. Quand une structure générique spécifie qu'un rapport doit contenir un titre, un résumé, une introduction, au moins deux chapitres et une conclusion, cela signifie seulement qu'un rapport, *une fois terminé*, devrait contenir tous ces éléments. Lorsque l'auteur commence à écrire, aucun de ces éléments n'est présent. Même lorsque le travail est bien avancé, de nombreux éléments peuvent encore être absents. Un éditeur qui utilise ce modèle doit donc nécessairement tolérer des documents non strictement conformes à la structure générique de leur classe ; il doit plutôt considérer la structure générique comme un moyen d'aider l'utilisateur dans la construction d'un document complexe.

D'autres applications au contraire peuvent rejeter un document qui ne serait pas strictement conforme à sa structure générique. C'est par exemple ce que font les compilateurs qui refusent d'engendrer le code pour un programme qui n'est pas syntaxiquement correct. C'est ce que pourrait faire une application documentaire pour un rapport qui n'aurait pas de résumé ou de titre.

Les constructeurs du modèle apportent une grande souplesse dans les structures génériques. Un choix, et plus encore une unité, peuvent représenter des éléments très différents. Le constructeur liste autorise la multiplication des éléments de même type. Combinés, ces deux constructeurs permettent n'importe quelle succession d'éléments de types différents. Bien sûr, cette souplesse peut être réduite là où c'est nécessaire et une structure générique peut limiter les choix et le nombre d'éléments des listes.

Une autre difficulté liée à l'utilisation de la structure dans le modèle de document réside dans le choix du niveau de structure. On pourrait envisager d'aborder le domaine linguistique pour extraire du texte même la structure du discours. Certaines études vont dans ce sens [Tazi], mais notre modèle exclut ce type de structure. Il ne prend en compte que la structure *logique* exprimée explicitement par l'auteur.

Cependant le niveau de structure du modèle n'est pas imposé. Chaque structure générique définit son propre niveau de structuration, adapté à la classe du document ou de l'objet, mais aussi aux traitements que le document doit subir. S'il s'agit seulement d'éditer et d'imprimer un document, une structure relativement simple suffit. S'il faut effectuer des traitements plus spécifiques, la structure doit représenter les types d'éléments sur lesquels les traitements devront agir. A titre d'exemple, la structure d'Edimath est suffisante pour imprimer des formules, mais une structure plus riche est nécessaire pour effectuer des calculs formels ou numériques sur des expressions mathématiques. Notre modèle autorise les deux types de structure.

5. UN LANGAGE POUR DEFINIR LES STRUCTURES GENERIQUES

Les structures génériques, qui constituent le fondement de notre modèle de document, sont exprimées dans un langage adapté. C'est ce langage de définition de structures génériques, appelé le langage S, que nous présentons dans cette section.

Chaque structure générique, qu'elle définisse une classe de documents ou d'objets, est spécifiée par une sorte de programme, écrit dans le langage S, qui est appelé un *schéma de structure*.

On verra plus loin que le langage S est compilé, et que, pour développer le compilateur, nous avons utilisé des outils spécifiques demandant une définition formelle du langage. C'est ce formalisme que nous utilisons dès maintenant pour présenter le langage S. C'est le même qui a été employé pour présenter la syntaxe de la représentation externe d'Edimath, et qui est décrit formellement dans l'annexe 1.

5.1. L'organisation générale d'un schéma de structure

Tout schéma de structure commence par le mot-clé STRUCTURE et se termine par le mot-clé END. Le mot-clé STRUCTURE est suivi du nom de la structure générique que le schéma définit (nom de la classe de document ou d'objet).

La définition du nom de la structure, est suivie par les déclarations du schéma de présentation par défaut, des attributs, des paramètres, les règles de structuration, les définitions des éléments associés et les définitions des unités exportées. Les déclarations des attributs et des paramètres sont optionnelles, de même que les définitions des éléments associés et des unités exportées. Chaque série de déclarations est introduite par un mot-clé : DEFPRES, ATTR, PARAM, STRUCT, ASSOC, UNITS.

```
SchemaStruct =  'STRUCTURE' IdentElem ';'
                'DEFPRES' IdentPres ';'
              [ 'ATTR' SuiteAttr ]
              [ 'PARAM' SuiteRegles ]
                'STRUCT' SuiteRegles
              [ 'ASSOC' SuiteRegles ]
              [ 'UNITS' SuiteRegles ]
                'END' .
IdentElem =      NAME .
```

5.2. La présentation par défaut

On a vu que plusieurs présentations différentes étaient possibles pour les documents et les objets d'une même classe. Le schéma de structure définit, pour une classe, une présentation préférentielle, dite présentation par défaut. Comme les structures génériques, les présentations sont décrites par des programmes, appelés *schémas de présentation*. C'est le nom du schéma de présentation par défaut qui est indiqué après le mot-clé DEFPRES. Les schémas de présentation sont écrits dans un langage spécifique, le langage P, présenté plus loin.

```
IdentPres =     NAME .
```

5.3. Les attributs

Si la structure générique comprend des attributs qui lui sont propres, ceux-ci sont déclarés après le mot-clé ATTR. Chaque attribut est défini par son nom, suivi du signe égal et, entre parenthèses, de la liste de toutes ses valeurs possibles. Il est probable que d'autres types d'attribut que ceux dont on peut énumérer les valeurs a priori seront nécessaires, notamment des attributs à valeur numérique. Ils ne sont pas (encore) inclus dans le langage.

Chaque valeur possible de l'attribut est un nom. Un attribut a au moins deux valeurs ; le nombre maximum de valeurs possibles est défini par le compilateur du langage S.

```
SuiteAttr = Attribut < Attribut > .
Attribut =  IdentAttr '=' '(' ValAttr < ',' ValAttr > ')' ';' .
IdentAttr = NAME .
ValAttr =   NAME .
```

5.4. Les paramètres

Un paramètre est un élément du document qui peut apparaître plusieurs fois dans le document, mais toujours avec la même valeur. Cette valeur ne peut être modifiée que de façon contrôlée, par certaines applications. Par exemple, pour une lettre circulaire, le nom du destinataire peut figurer dans la partie adresse et dans la formule de politesse finale. Une valeur ne peut lui être affectée que par l'application de publipostage.

Toutes les classes de documents n'utilisent pas des paramètres, mais si le schéma comprend des paramètres, ceux-ci sont déclarés après le mot-clé PARAM. Chaque déclaration de paramètre est constituée de la même façon qu'une déclaration d'élément structuré (voir plus loin).

5.5. Les éléments structurés

Les règles de définition des éléments structurés sont obligatoires : elles constituent l'essentiel d'un schéma de structure, puisqu'elles définissent la structure des différents types d'éléments qui interviennent dans un document ou un objet de la classe définie par le schéma.

La première règle de structuration après le mot STRUCT doit être celle qui définit la structure de la classe dont le nom apparaît dans la première instruction (STRUCTURE). C'est la règle racine du schéma, celle qui définit la racine de l'arbre d'un document ou d'un objet de la classe.

Chaque règle est constituée d'un nom (le nom du type d'élément dont on définit la structure) suivi du signe égal et d'une définition de structure. Si des valeurs d'attribut doivent être systématiquement attachées à ce type d'élément, la définition de structure est suivie du mot-clé WITH et de la liste des attributs à valeur imposée. La règle se termine par un point-virgule.

```
SuiteRegles = Regle < Regle > .
Regle =       IdentElem '=' DefAvecAttr ';' .
DefAvecAttr = Definition [ 'WITH' SuiteAttrFixes ] .
```

La liste des attributs à valeur imposée est constituée d'une suite de couples attribut-valeur, séparés par des virgules. Chaque couple comprend le nom de l'attribut dont la valeur est imposée pour ce type d'élément, et le nom de sa valeur imposée, les deux noms étant séparés par un signe égal.

```
SuiteAttrFixes = AttrFixe < ',' AttrFixe > .
AttrFixe =       IdentAttr '=' ValAttr .
```

5.6. Les définitions de structure

La structure d'un type d'élément peut être soit un simple type de base, soit un type construit. Il est fréquent de trouver des structures semblables en plusieurs endroits d'un document. Par exemple, les contenus du résumé, de l'introduction et d'une section peuvent avoir la même structure, celle d'une suite de paragraphes. Dans ce cas on définit une seule fois la structure commune, avec un nom commun (Suite_de_paragraphes) et on indique que chacun des types d'élément particuliers a cette structure, de la façon suivante :

```
Résumé = Suite_de_paragraphes;
Introduction = Suite_de_paragraphes;
Contenu_de_section = Suite_de_paragraphes;
```

Le signe = signifie "a la même structure que".

Si le type d'élément défini est un simple type de base, il est indiqué par l'un des mots-clés TEXT, GRAPHICS, SYMBOL ou PICTURE. Dans le cas où le type est libre (c'est une unité), il est indiqué par le mot-clé UNIT. Ainsi, on spécifiera le nom de l'auteur d'un

document comme TEXT, et le contenu d'un paragraphe comme une suite d'UNITs, ce qui permettra d'inclure dans les paragraphes des chaînes de caractères, des symboles, et des objets de natures diverses, comme des formules mathématiques par exemple. Une unité offre le choix entre trois catégories d'éléments :

- un des types de base : texte, élément graphique, symbole, image,
- un objet défini par un schéma de structure choisi librement parmi les schémas disponibles ; dans ce cas l'élément a le type défini par la première règle (la règle racine) du schéma choisi,
- un élément dont le type est choisi parmi les types définis comme unités exportables dans la section UNITS du le schéma de structure de la classe du document.

Si le type d'élément défini est un type construit, on utilise l'un des constructeurs liste, agrégat, choix, ou référence. Dans ce cas, la définition commence par un mot-clé identifiant le constructeur. Ce mot-clé est suivi d'une partie spécifique à chaque constructeur.

```
Definition = TypeDeBase / Constr / Element .
TypeDeBase = 'TEXT' / 'GRAPHICS' / 'SYMBOL' / 'PICTURE' / 'UNIT' .
Element =    IdentElem [ '=' Definition ] .
Constr =     'LIST' [ '[' min '..' max ']' ] 'OF'
                                          '(' DefAvecAttr ')' /
             'BEGIN' SuiteDef 'END' /
             'CASE' 'OF' SuiteDef 'END' /
             'REFERENCE' '(' IdentElem ')' .
```

• Liste : le type d'élément défini est constitué d'une liste d'éléments, tous du même type. La définition de la liste est formée du mot-clé LIST suivi, entre crochets, des nombres minimum et maximum d'éléments de la liste, séparés par deux points, et, après le mot-clé OF, de la définition entre parenthèses du type des éléments qui doivent constituer la liste. Les nombres minimum et maximum d'éléments sont optionnels, mais si l'un des deux nombres est spécifié, l'autre doit l'être également. Dans le cas où l'un des deux nombres est indéterminé, il est remplacé par une étoile.

```
          'LIST' [ '[' min '.' '.' max ']' ] 'OF' '(' DefAvecAttr ')'
min =     Entier / '*' .
max =     Entier / '*' .
Entier = NUMBER .
```

• Agrégat : le type d'élément défini est constitué d'un ensemble d'éléments, chacun ayant un type fixé. Dans la définition d'un agrégat, le mot-clé BEGIN est suivi de la liste des définitions des types des éléments devant constituer l'agrégat. Cette liste se termine avec le mot-clé END. Les définitions de type des constituants sont séparées les unes des autres par des points-virgules.

```
'BEGIN' SuiteDef 'END' SuiteDef = DefAvecAttr ';' < DefAvecAttr
';' > .
```

• Choix : le type de l'élément peut être choisi parmi un ensemble de types possibles. Les mots-clés CASE et OF sont suivis de la liste des définitions des types possibles, séparés les unes des autres par des points-virgules. Cette liste se termine par le mot-clé END'.

```
'CASE' 'OF' SuiteDef 'END'
```

• Référence : le type de l'élément défini est une référence à un élément d'un type donné. Ce type est indiqué entre parenthèses après le mot-clé REFERENCE.

```
'REFERENCE' '(' IdentElem ')'
```

5.7. Les importations

Grâce au constructeur UNIT, il est possible, lors de l'édition d'un document, d'utiliser des classes définies par d'autres schémas de structure, qui sont alors indiqués dynamiquement par l'utilisateur, au fur et à mesure de ses besoins. On peut aussi, dans le schéma de structure, imposer une nature (une classe) particulière à certains types d'éléments. Dans ce cas, ces classes sont simplement désignées par leur nom, lorsqu'elles sont utilisées dans les définitions de type d'éléments structurés : si un nom de type n'est pas défini dans le schéma de structure, c'est qu'il s'agit du nom d'une structure définie par un autre schéma de structure.

5.8. Les éléments associés

Si des éléments associés sont nécessaires, ils doivent être déclarés dans la section spécifique du schéma de structure, introduite par le mot-clé ASSOC. Chaque type d'élément associé est spécifié comme n'importe quel autre élément structuré. Cependant, ces types ne doivent apparaître dans aucune autre règle du schéma, sauf éventuellement dans des règles REFERENCE.

5.9. Les unités exportées

La section UNITS du schéma de structure contient les déclarations des types d'éléments qui pourront être exportés vers les objets inclus dans les documents de la classe définie par le schéma. Tout comme les éléments associés, ces types d'éléments sont définis comme n'importe quel autre type d'élément structuré. Ils peuvent être utilisés dans les autres types d'éléments du schéma, mais ils peuvent aussi n'être utilisés par aucune règle du schéma.

6. QUELQUES EXEMPLES

Pour illustrer à la fois les principes du modèle et la syntaxe du langage S, on donne ici deux exemples de schémas de structure, l'un qui définit une classe de documents, l'autre, une classe d'objets.

6.1. Une classe de documents : les articles

Dans cet exemple, on décrit une structure possible pour des articles publiés par une revue scientifique. Les textes entre accolades sont des commentaires.

```
STRUCTURE Article;   { Ce schéma définit la classe Article }
DEFPRES ArticleP;    { Le schéma de présentation par défaut est ArticleP }

ATTR                 { Définition des attributs }
   Langue = (Français, Anglais);  { Un seul attribut est défini, la langue }

STRUCT               { Définition de la structure générique }
   Article = BEGIN   { L'article a une structure d'agrégat }
             Titre = BEGIN    { Le titre a une structure d'agrégat }
                     Titre_français = Text WITH Langue=Français;
                     Titre_anglais =  Text WITH Langue=Anglais;
                     END;
             Auteurs = LIST OF (Auteur =  { Un ou plusieurs auteurs }
                                  BEGIN
                                  Nom_Auteur = Text;
                                  Curriculum = Paragraphes;
                                  Adresse =    Text;
                                  END
```

```
                    );
        Domaine = Text;
        Mots_Clés = Text;
          { L'éditeur du journal présente l'article par un court texte
            d'introduction, en français et en anglais }
        Présentation = BEGIN
                       Prés_Français = Paragraphes WITH
                                       Langue=Français;
                       Prés_Anglais = Paragraphes WITH
                                       Langue=Anglais;
                       END;
        Corps = Sections;
        Annexes = LIST OF (Annexe =
                           BEGIN
                           Titre_Annexe = Text;
                           Contenu_Annexe = Paragraphes;
                           END
                          );
        END;                          { Fin de l'agrégat Article }

Sections = LIST [2..*] OF (Section =    { Il y a au moins 2 sections }
                           BEGIN
                           Titre_Section = Text;
                           Contenu_Section =
                                  BEGIN
                                  Paragraphes;
                                  Sections;
                                  { Sections de niveau inférieur }
                                  END;
                           END
                         );

 Paragraphes = LIST OF (Paragraphe =
                    LIST OF (CASE OF
                        Enumération = LIST [2..*] OF
                                        (Item = Paragraphes);
                        Formule_isolée = Formule;
                        UNIT;
                        Réfer;
                        END )
```

```
                    );

ASSOC          { Définition des éléments associés }

   Figure =    BEGIN
               Légende_figure = Text;
               Illustration = UNIT;
               END;

   Citation_biblio = CASE OF
                        Ref_Article =
                           BEGIN
                           Auteurs_Bib =   Text;
                           Titre_Article = Text;
                           Ouvrage =       Text;
                           Numéros_Pages = Text;
                           Date =          Text;
                           END;
                        Ref_Livre =
                           BEGIN
                           Auteurs_Bib; { Défini plus haut }
                           Titre_Livre =   Text;
                           Editeur =       Text;
                           Date;         { Défini plus haut }
                           END;
                     END;

   Note =  Paragraphes;

UNITS          { Les éléments exportés vers les objets inclus }

   Réfer =     CASE OF       { Toutes les références possibles }
               Ref_note =     REFERENCE (Note);
               Ref_biblio =   REFERENCE (Citation_biblio);
               Ref_figure =   REFERENCE (Figure);
               Ref_formule = REFERENCE (Formule_isolée);
               END;

END
```

Ce schéma est très complet puisqu'il va jusqu'à définir les paragraphes et les citations bibliographiques. Ce type d'élément pourrait aussi bien être défini par d'autres schémas de structure, comme c'est le cas de la classe Formule, explicitement référencée ici. Toutes sortes d'autres éléments peuvent être introduits dans un article, puisqu'un paragraphe peut contenir des unités quelconques. De même, les illustrations peuvent être de différentes classes et l'utilisateur choisira, pour chaque figure, la classe la mieux adaptée.

Les éléments Figure, Citation_biblio et Note sont des éléments associés. Ils ne sont utilisés que dans des instructions REFERENCE.

Des renvois de toutes sortes (Réfer) peuvent être utilisés dans les paragraphes, mais aussi dans des objets inclus dans l'article, puisqu'ils sont définis comme unités exportées (UNITS).

On remarquera que le langage S permet la définition de structures récursives, comme les sections : une section peut contenir des sections (qui sont alors de niveau inférieur). Les paragraphes sont également des éléments récursifs, puisqu'un paragraphe peut contenir une énumération dont chaque élément (Item) est constitué de paragraphes.

On remarquera également la souplesse du langage qui permet de définir les types d'éléments avant ou après leur utilisation, ou même dans l'instruction où ils sont utilisés.

6.2. Une classe d'objets : les formules d'Edimath

L'exemple ci-dessous définit la classe Formule utilisée notamment dans les documents Article. Cette classe représente les formules mathématiques avec une structure très proche de celle d'Edimath. Elle n'utilise aucune autre classe et ne définit pas d'élément associé ni d'unité exportée.

```
STRUCTURE Formule;
DEFPRES FormuleP;

ATTR
   alphabet =  (ASCII, grec, scientifique);

STRUCT
   Formule =     Expression;
   Expression = LIST OF (Construction = CASE OF
                  Text;          { Simple chaîne de caractères }
                  Indice = Expression;
                  Exposant = Expression;
```

```
        Fraction =
              BEGIN
              Numérateur = Expression;
              Dénominateur = Expression;
              END;
        Racine = Expression;
        Intégrale =
              BEGIN
              Borne_Inférieure = Expression;
              Borne_Supérieure = Expression;
              END;
        Triple =
              BEGIN
              Expression_Princ = Expression;
              Expression_Inf = Expression;
              Expression_Sup = Expression;
              END;
        Vecteur = LIST [2..*] OF (Elément = Expression);
        Bloc_Parenthèse =
              BEGIN
              Ouvrant = SYMBOL;
              Contenu = Expression;
              Fermant = SYMBOL;
              END;
        END);          { Fin du Choix Construction }
END                    { Fin du schéma de structure }
```

7. LA PRESENTATION DES DOCUMENTS

Grâce au modèle que nous avons retenu, la présentation des documents est clairement séparée de leur structure et de leur contenu. Après avoir présenté la structure des documents, nous détaillons maintenant les principes mis en œuvre pour leur présentation.

La notion de présentation recouvre à la fois ce qu'on appelle la mise en page, la composition et la maquette d'un document. Dans certains systèmes de manipulation de documents structurés, ce que nous appelons *présentation* est appelé *décompilation* ou encore *paragraphage*. Nous préférons le terme présentation, qui décrit mieux ce qu'il désigne, puisque nous nous intéressons aux documents en général. En effet, la décompila-

tion s'applique plutôt aux programmes, pour lesquels le mot compilation a un sens très particulier ; mais les documents qui nous intéressent ne sont pas compilés. D'autre part ce sont des documents relativement complexes, dont la mise en page ne se borne pas à la simple construction de paragraphes ; le terme paragraphage est donc également mal adapté.

7.1. Une présentation à deux niveaux

Le lien entre structure et présentation est clair : l'organisation d'un document peut être mise à profit pour effectuer sa présentation, puisque la présentation est justement faite pour mettre en évidence l'organisation du document. Mais elle est également dépendante de l'appareil utilisé pour la restitution. Certains effets de présentation, comme les changements de police ou de corps notamment, ne peuvent pas être obtenus sur toutes les imprimantes et sur tous les écrans. C'est pourquoi nous avons retenu une approche à deux niveaux, où la présentation est d'abord décrite en termes abstraits, sans tenir compte d'aucun appareil particulier, fût-il virtuel, puis elle est réalisée en tenant compte des limites d'un appareil donné.

La présentation est donc décrite uniquement en fonction de la structure des documents et de l'image que l'on souhaite obtenir avec un appareil idéal. Pour cette raison, dans la description de la présentation, on ne fait référence à aucune caractéristique d'appareil : il s'agit d'une *présentation abstraite* qui peut être concrétisée sur des appareils différents.

Il s'agit aussi d'une *présentation générique* qui décrit l'aspect d'une classe de documents ou d'objets. Cette présentation générique doit être appliquée à des occurrences de documents ou d'objets, chacun étant bien sûr conforme à sa structure générique, mais avec toutes les tolérances évoquées plus haut : éléments manquants, éléments construits avec d'autres structures génériques, etc...

Pour conserver l'homogénéité entre documents et objets, la présentation doit pouvoir se décrire avec un ensemble unique d'outils, qui permettent la mise en page d'un gros document aussi bien que la composition d'un objet comme un graphique ou une formule. Cette unicité des outils de description de la présentation est rarement atteinte dans les systèmes de production de documents. L'approche traditionnelle s'intéresse plus aux documents qu'aux objets. Elle se fonde sur les données typographiques habituelles, comme les positions des marges, les retraits, les espaces verticaux, la justification, les changements de police, etc... Cette approche est utilisée dans des systèmes qui traitent essentiellement (voire exclusivement) du texte et qui, lorsqu'ils traitent d'autres types d'information, utilisent d'autres principes de présentation pour l'information non-textuelle.

Scribe [Reid 80] illustre bien cette approche. La présentation des *environnements* est décrite dans la base de données, et peut être modifiée par l'utilisateur, dans ces termes : marges, espace vertical avant ou après l'élément, interligne dans l'élément, justification, etc... En revanche, la présentation des formules mathématiques, qui ont été ajoutées *a posteriori* dans Scribe, n'est pas modifiable par l'utilisateur. De toute façon, les paramètres de présentation que l'utilisateur peut contrôler ne permettraient pas de modifier l'image d'une formule. Interleaf [Morris 85], de ce point de vue, est tout à fait comparable à Scribe, puisqu'il offre à l'utilisateur la possibilité de contrôler, pour le texte, le même type de paramètres de présentation, grâce aux *propriétés* attachées aux éléments. Mais en mode graphique, s'il est effectivement possible de changer l'aspect d'une figure, cela se fait avec des paramètres différents.

7.2. Les boîtes

Pour assurer l'homogénéité des outils, nous avons basé toute la présentation, pour les documents comme pour les objets qu'ils contiennent, sur la notion de *boîte*, telle qu'elle a été mise en œuvre dans TEX [Knuth 84] ou dans Edimath. Cette notion a d'ailleurs été utilisée dans d'autres systèmes, et notamment des systèmes considérant les objets non-textuels, comme par exemple GTX [Nanard 84].

A chaque élément du document on fait correspondre une boîte, qui est le rectangle englobant l'élément sur le support d'affichage (écran ou feuille de papier) ; le contour de ce rectangle n'est normalement pas visible. Les côtés de la boîte sont parallèles aux bords de l'écran ou de la feuille de papier. A titre d'exemple, une boîte est associée à une chaîne de caractères, à une ligne de texte, à une page, à un paragraphe, à un titre, à une formule mathématique, ou encore à une expression à l'intérieur d'une formule (comme dans Edimath), ou à une cellule d'un tableau.

Quel que soit l'élément auquel elle se rapporte, chaque boîte possède quatre côtés et quatre axes, que nous désignons de la façon suivante (voir figure 3.2) :

- `Top` : le côté supérieur,
- `Bottom` : le côté inférieur,
- `Left` : le côté gauche,
- `Right` : le côté droit,
- `VMiddle` : l'axe vertical passant par le centre de la boîte,
- `HMiddle` : l'axe horizontal passant par le centre de la boîte,
- `VRef` : l'axe de référence vertical,
- `HRef` : l'axe de référence horizontal.

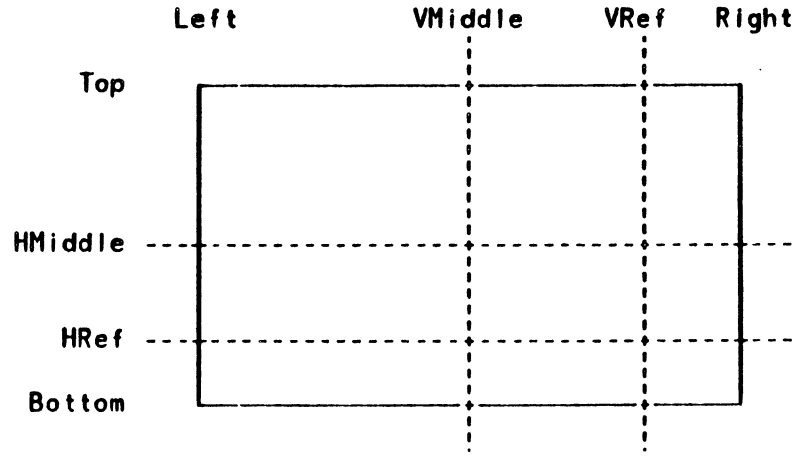


Figure 3.2 : Les côtés et axes des boîtes.

Le rôle principal des boîtes est de dimensionner et de positionner les images des différents éléments d'un document les uns par rapport aux autres, sur le support de restitution. Pour cela on définit, entre les boîtes des différents éléments, des relations qui donnent les dimensions et les positions relatives de ces boîtes.

On distingue trois types de boîtes :

- les boîtes associées aux éléments du document,
- les boîtes de présentation,
- les boîtes de mise en page.

Les boîtes associées aux éléments du document sont celles qui correspondent à chacun des éléments (de base ou structurés) de la structure logique du document. Une telle boîte contient (délimite sur le support d'affichage) tout le contenu de l'élément auquel elle est associée. Toutes ces boîtes forment une structure arborescente, identique à celle des éléments structurés auxquels elles correspondent. Comme dans Edimath, cet arbre exprime les relations d'inclusion entre les boîtes : une boîte englobe toutes les boîtes de son sous-arbre. En revanche il n'y a pas de règle prédéfinie pour les positions relatives des boîtes englobées. Si elles sont de même niveau, elles peuvent se chevaucher, être contiguës, ou disjointes. Les règles de la présentation générique précisent ces positions relatives.

Les **boîtes de présentation** représentent des éléments qui ne figurent pas dans la structure logique du document, mais qui sont ajoutées pour les besoins de la présentation. Ces boîtes sont reliées aux éléments de la structure, dont elles permettent de mieux faire apparaître le type. Elles sont par exemple utilisées pour ajouter la chaîne de caractères 'Résumé :' avant le résumé dans l'image d'un rapport, ou pour représenter la barre de fraction dans une formule, ou encore pour faire apparaître les champs fixes d'un formulaire. Ces éléments, en effet, n'ont pas de rôle dans la structure logique du

document. Si un élément est de type résumé, il n'est pas nécessaire de faire figurer, dans la structure, le mot 'Résumé'. De même, si un élément est du type fraction et qu'il comporte deux autres éléments de type numérateur et dénominateur, la barre n'a pas d'utilité structurelle. En revanche, ces éléments de présentation sont importants pour le lecteur du document restitué ou pour l'utilisateur d'un éditeur, c'est pourquoi ils doivent figurer dans l'image du document. C'est la présentation générique qui spécifie les boîtes de présentation à ajouter, en indiquant leur contenu (un élément de base dont la valeur est précisée) et la position qu'elles doivent prendre dans l'arbre des boîtes. Lors de l'édition, ces boîtes ne peuvent pas être modifiées par l'utilisateur.

Les boîtes de mise en page sont des boîtes créées implicitement par les règles de mise en page. Ces règles indiquent comment le contenu d'un élément structuré doit être découpé en lignes et les lignes regroupées en colonnes ou en pages. A la différence des boîtes de présentation, ces boîtes lignes, pages ou colonnes ne dépendent pas de la structure logique du document, mais plutôt des contraintes physiques du support de restitution : taille des caractères, largeur et hauteur de l'écran ou de la feuille de papier. La présentation générique n'indique que les règles qui doivent être suivies pour la construction de ces boîtes, mais ne les crée pas explicitement, comme elle le fait pour les boîtes de présentation.

7.3. Les vues et la visibilité

Selon les traitements que l'on veut effectuer sur un document, on peut souhaiter voir ce document de différentes façons. Pour cette raison, il est possible de définir plusieurs *vues* pour un même document, ou mieux, pour tous les documents d'une même classe. Une vue n'est pas une présentation différente du document, mais plutôt un filtre qui permet de n'afficher que certaines parties du document. Par exemple, on peut souhaiter ne voir que les titres de chapitres et de sections, mais les voir tous, pour pouvoir se déplacer rapidement dans le document ; on a alors besoin d'une vue "table des matières". On peut aussi souhaiter ne voir que les formules mathématiques d'un document, si on s'intéresse plus au raisonnement mathématique lui-même qu'à son texte de présentation. Il faut pour cela une vue "mathématique" du document.

Les vues, comme la présentation, sont basées sur la structure générique. Chaque classe de document, et même chaque présentation générique, peut donc être munie des vues qui lui sont spécifiquement utiles. Pour chaque vue, on définit la *visibilité* des éléments, qui indique si les éléments doivent être présentés à l'utilisateur ou non. La visibilité est calculée en fonction du type des éléments et/ou de leur position hiérarchique dans la structure du document. Ainsi, pour une table des matières, tous les éléments "Titre de chapitre" et "Titre de section" sont rendus visibles, le niveau hiérarchique pouvant être

utilisé pour rendre les titres de section invisibles au-dessous d'un certain niveau seuil. En faisant varier ce seuil, on peut faire varier la granularité de la vue. Dans la vue "mathématique", on rend visibles les seuls éléments du type formule, quel que soit leur niveau hiérarchique.

Les vues étant surtout utilisées pour donner une image synthétique du document, il est nécessaire d'adapter la présentation des éléments selon la vue dans laquelle ils apparaissent : on ne change pas de page pour chaque titre de chapitre dans la table des matières. Les présentations génériques tiennent donc compte des vues possibles et permettent de définir, pour chaque type d'élément, des présentations différentes selon les vues.

Les vues sont également utilisées, pendant l'édition des documents, pour afficher les éléments associés. On peut ainsi avoir, en plus de la vue principale du document, une vue "notes" et une vue "figures" qui contiennent chacune les éléments associés du type note ou figure. De cette façon, il est possible de voir (donc d'éditer) en même temps le texte qui fait référence à ces éléments et les éléments eux-mêmes, même s'ils doivent être séparés au moment de l'impression. On peut, grâce aux vues, retrouver la notion de galée de Mint [Hibbard].

7.4. La numérotation

De nombreux éléments sont numérotés dans les documents : les chapitres, les sections, les formules, les théorèmes, les notes, les figures, les références bibliographiques, les exercices, les exemples, les lemmes... Avec la notion de structure que nous avons retenue, tous ces numéros sont redondants avec la structure même du document. Ils ne constituent qu'un moyen de rendre plus visible la structure du document. Ils font donc partie de la présentation du document et sont calculables à partir de la structure. La structure en tant que telle ne comporte pas de numéros, elle définit seulement des positions structurelles relatives entre éléments, donc des relations d'ordre entre éléments.

Si le corps d'un document est défini comme une suite d'au moins deux chapitres :

```
Corps = LIST [2..*] OF Chapitre ;
```

la suite définie par le constructeur liste est ordonnée et on peut attribuer comme numéro à chaque chapitre son rang dans la liste Corps. Tous les éléments constituant des listes dans la structure d'un document sont donc numérotables, mais ce ne sont pas les seuls. La structure arborescente induite par les constructeurs agrégat, liste, et choix, à l'exclusion des références, permet de définir un ordre total entre tous les éléments du document à l'exclusion des éléments associés. Il est donc possible de définir une numérotation qui

utilise cet ordre, en filtrant selon leur type les éléments qui doivent être pris en compte. De cette façon, on peut donc choisir de numéroter ensemble, dans une même séquence de numéros, tous les théorèmes et les lemmes d'un chapitre, même s'ils ne font pas partie d'une même construction liste ; mais on peut aussi, en changeant le filtre, les numéroter séparément : une suite de numéros pour les théorèmes, une autre pour les lemmes.

Les éléments associés posent un problème particulier, puisqu'ils ne font pas partie de la structure principale du document, mais qu'ils y sont attachés uniquement par des références, qui précisément rompraient l'ordre total obtenu dans le document. Or ces éléments associés sont très souvent numérotés, justement parce que le numéro est un moyen efficace de visualiser la référence. Pour résoudre ce problème, on regroupe les éléments associés par types, et pour chaque type on définit une construction liste qui regroupe (et ordonne) tous les éléments associés de même type. Les éléments associés peuvent ainsi être numérotés par type.

Etant calculés à partir de la structure et uniquement pour les besoins de la présentation, les numéros sont donc des éléments de présentation, décrits par des boîtes de présentation, tout comme la barre de fraction ou le mot 'Résumé'. Ils se différencient toutefois de ceux-ci par le fait que leur contenu est variable d'une occurrence à l'autre d'un même type, alors que les barres de toutes les fractions sont des traits horizontaux et que le même mot 'Résumé' apparaît en tête des résumés de tous les documents de la classe.

7.5. Les paramètres de présentation

Les principaux paramètres qui déterminent la présentation des documents sont les *positions* et *dimensions* des boîtes, et la *mise en évidence* et la *taille* de leur contenu. A partir de ces quelques paramètres, et de quelques autres de moindre importance, il est possible, pour les parties textuelles, de représenter les paramètres usuels de la typographie. Pour les éléments non-textuels, qui sont des éléments bidimensionnels, et non pas linéaires comme le texte, ces mêmes paramètres permettent de décrire leur géométrie.

Comme on l'a déjà vu, les positions des boîtes respectent toujours la règle d'englobement : une boîte de l'arbre englobe toutes les boîtes de niveau inférieur qui lui sont attachées. Les paramètres de positionnement permettent de préciser la position de chaque boîte par rapport à la boîte englobante ou aux boîtes sœurs (boîtes attachées directement, dans l'arbre des boîtes, à la même boîte englobante).

Les paramètres de présentation permettent également de contrôler les dimensions des boîtes. Les dimensions d'une boîte peuvent dépendre soit de son contenu, soit de son environnement (les boîtes sœurs et la boîte englobante), chaque dimension (hauteur ou largeur) pouvant être définie indépendamment de l'autre.

Grâce aux paramètres de position et de dimension des boîtes, on peut effectuer ce qui est habituellement fait en typographie par des changements de marges, des changements de longueur de ligne, des sauts d'espace vertical ou horizontal. On peut aussi effectuer l'équivalent des alignements, des centrages, etc...

Au contraire de la position ou de la dimension, la mise en évidence et la taille ne concernent pas la boîte elle-même (le rectangle délimitant l'élément), mais son contenu. Ces paramètres indiquent les attributs typographiques qui doivent être appliqués au texte contenu dans la boîte, et, par extension, à tous les éléments de base. Ainsi la mise en évidence permet d'obtenir des changements de style de caractères, de l'italique ou du romain, du gras ou du maigre, quand il s'agit de caractères. Mais pour le graphique, ce paramètre se traduit par l'épaisseur du trait ou son style (continu, pointillé, tirets...). La taille, elle, représente le corps de caractères ou la grosseur des éléments graphiques.

8. UN LANGAGE POUR DECRIRE LA PRESENTATION

Une présentation générique définit les valeurs des paramètres de présentation (ou le mode de calcul de ces valeurs) pour une structure générique, ou plus précisément pour tous les types d'éléments et tous les attributs définis dans cette structure générique. Cette définition des paramètres de présentation se fait à l'aide d'un langage, appelé le langage P. Un programme écrit dans ce langage, donc une présentation générique exprimée en P, est appelé un *schéma de présentation*. Nous décrivons ici la syntaxe et la sémantique du langage, en utilisant le même méta-langage que pour la définition du langage S.

8.1. L'organisation d'un schéma de présentation

Un schéma de présentation commence par le mot PRESENTATION et se termine par le mot END. Le mot PRESENTATION est suivi du nom de la structure générique dont la présentation est définie par le schéma. Ce nom doit être le même que celui qui suit le mot-clé STRUCTURE dans le schéma de structure auquel est associé le schéma de présentation.

Après cette déclaration du nom de structure, apparaissent, dans l'ordre :

- les déclarations
  - des vues,
  - des compteurs de numérotation,
  - des constantes de présentation,
  - des variables,
- les règles de présentation par défaut.

- les définitions des boîtes de présentation et de mise en page,
- les règles de présentation des éléments structurés,
- les règles de présentation des attributs.

Chacune de ces sections est introduite par un mot-clé qui est suivi d'une suite de déclarations. Toutes ces sections sont optionnelles, sauf celle des règles de présentation par défaut et celle des règles de présentation des éléments structurés.

```
SchemaPres = 'PRESENTATION' IdentType ';'
             [ 'VIEWS' SuiteVues ]
             [ 'COUNTERS' SuiteCompteurs ]
             [ 'CONST' SuiteConst ]
             [ 'VAR' SuiteVar ]
               'DEFAULT' SuiteReglesVues
             [ 'BOXES' SuiteBoites ]
               'RULES' SuitePresent
             [ 'ATTRIBUTES' SuitePresAttr ]
               'END' .
IdentType = NAME .
```

8.2. Les vues

Chacune des vues possibles pour la classe doit être déclarée dans le schéma de présentation. On a vu que les règles de présentation d'un type d'élément pouvaient varier suivant la vue dans laquelle l'élément apparaît ; c'est le nom de la vue qui sera utilisé pour désigner une vue dans les règles de présentation (voir l'instruction IN). La définition du contenu de la vue est dispersée à travers les règles de présentation attachées aux différents types d'éléments et d'attributs. La section VIEWS est simplement constituée de la suite des noms des vues séparés par des virgules et terminée par un point-virgule.

On peut ne déclarer aucune vue ; dans ce cas il y aura néanmoins une (et une seule) vue possible, qui n'aura pas de nom. Si plusieurs vues sont déclarées, celle dont le nom figure en tête de liste est considérée comme la vue principale, c'est à dire celle à laquelle se rapportent toutes les règles de présentation qui ne sont pas explicitement précédées d'une indication de vue (voir l'instruction IN).

```
            'VIEWS' SuiteVues
SuiteVues = IdentVue < ',' IdentVue > ';' .
IdentVue = NAME .
```

Exemple : si, lors de l'édition d'un rapport, on désire une vue pour la table des matières et une autre pour toutes les formules, en plus de la vue principale qui montre le document dans son entier, on déclare dans un schéma de présentation de la classe des rapports :

```
VIEWS
    Texte_intégral, Table_des_matières, Formules;
```

Le contenu de ces vues est précisé dans les règles de présentation du schéma.

8.3. Les compteurs

A chaque type de numérotation le schéma de présentation associe un compteur. Tous les compteurs, donc tous les types de numérotation, utilisés dans le schéma doivent être déclarés après le mot-clé COUNTERS.

Chaque déclaration de compteur est constituée d'un nom identifiant le compteur, suivi de ':' et de la fonction de comptage à appliquer au compteur. La déclaration d'un compteur se termine par un point-virgule.

La fonction de comptage indique le mode de calcul de la valeur du compteur. Deux fonctions de comptage sont disponibles. La première s'applique au comptage des éléments de liste : elle affecte au compteur le rang de l'élément dans la liste. Plus précisément, la fonction

```
RANK OF IdentType
```

indique que lorsqu'un élément crée, par une règle de création (voir plus loin les instructions `Create`), une boîte de présentation contenant la valeur du compteur, cette valeur est le rang de l'élément créateur, s'il est de type `IdentType`, sinon le rang du premier élément de type `IdentType` qui englobe l'élément créateur dans la structure logique du document.

La deuxième fonction de comptage permet de compter les occurrences d'un certain type d'élément dans un contexte précisé. L'instruction

```
SET n ON Type1 ADD m ON Type2
```

signifie que, lorsqu'on parcourt le document du début à la fin, dans l'ordre induit par la structure arborescente, on affecte au compteur la valeur n chaque fois qu'on rencontre un élément de type Type1, quelle que soit la valeur courante du compteur, et on ajoute la valeur m à la valeur courante du compteur chaque fois qu'on rencontre un élément de type Type2.

La définition formelle des déclarations de compteurs s'écrit :

```
                'COUNTERS' SuiteCompteurs
SuiteCompteurs = Compteur < Compteur > .
Compteur =       IdentCompteur ':' FonctCompteur ';' .
IdentCompteur =  NAME .
FonctCompteur = 'RANK' 'OF' IdentType /
                'SET' ValeurCompt 'ON' IdentType
                'ADD' ValeurCompt 'ON' IdentType .
ValeurCompt =    NUMBER .
```

Exemple : si le corps d'un chapitre est défini, dans le schéma de structure, comme une séquence de sections :

```
Corps_Chapitre = LIST OF (Section =
                              BEGIN
                              Titre_Section = Text;
                              Corps_Section = Paragraphes;
                              END
                         );
```

le compteur de sections sera déclaré :

```
CptSection : RANK OF Section;
```

Pour numéroter des formules chapitre par chapitre, le compteur de formules sera déclaré :

```
CptFormule : SET 0 ON Chapitre ADD 1 ON Formule;
```

et l'affichage du numéro de section devant chaque titre de section sera obtenu par une règle CreateBefore (voir plus loin), attachée au type Titre_Section, qui créera une boîte de présentation dont le contenu sera la valeur du compteur CptSection (voir l'instruction Content). De même, l'affichage du numéro de formule le long de la marge droite, en face de chaque formule, sera obtenu par une instruction CreateAfter, attachée au type Formule, qui créera une boîte de présentation dont le contenu sera la valeur du compteur CptFormule.

8.4. Les constantes de présentation

Les constantes de présentation permettent de définir le contenu des boîtes de présentation. Ce contenu est utilisé dans les définitions de variables et dans la règle `Content`. Les seules constantes de présentation qui peuvent être utilisées sont des chaînes de caractères, des symboles mathématiques, des éléments graphiques, et des images, c'est à dire des éléments de base.

Les constantes peuvent être définies directement dans les variables ou les boîtes de présentation (règle `Content`) qui les utilisent. Mais on a intérêt à déclarer une seule fois, dans la section des déclarations de constantes, les constantes qui sont utilisées dans plusieurs variables ou boîtes. Chaque constante ainsi déclarée porte un nom, qui permet de la désigner lorsqu'on l'utilise, un type (l'un des quatre types de base) et une valeur (une chaîne de caractères ou un caractère unique pour les symboles et les éléments graphiques).

Après le mot-clé `CONST` apparaissent toutes les déclarations de constantes. Chacune est constituée du nom de la constante, d'un signe égal, d'un mot-clé représentant son type (`Text`, `Symbol`, `Graphics` ou `Picture`) et de la chaîne représentant sa valeur ; elle se termine par un point-virgule.

```
                'CONST' SuiteConst
SuiteConst =    Const < Const > .
Const =         IdentConst '=' TypeConst ValeurConst ';' .
IdentConst =    NAME .
TypeConst =    'Text' / 'Symbol' / 'Graphics' / 'Picture' .
ValeurConst = STRING .
```

Dans le cas d'un symbole ou d'un élément graphique, la valeur ne comporte qu'un seul caractère entre apostrophes qui indique la forme de l'élément qui doit être tracé dans la boîte dont la constante est le contenu :

Graphique :

`'C'` un cercle inscrit dans la boîte : ○,
`'R'` un rectangle qui est le contour de la boîte : □,
`'t'` un trait horizontal : le côté supérieur de la boîte : ¯,
`'h'` un trait horizontal au milieu de la boîte, de la longueur de la boîte : −,
`'b'` un trait horizontal : le côté inférieur de la boîte : _,
`'l'` un trait vertical : le côté gauche de la boîte : |,
`'v'` un trait vertical au milieu de la boîte, de la hauteur de la boîte : |,
`'r'` un trait vertical : le côté droit de la boîte : |,
`' '` un élément transparent :

Symbole:

'r' un radical de la dimension de la boîte : $\sqrt{}$,
'i' une intégrale simple de la hauteur de la boîte : $\int$,
'd' une intégrale double de la hauteur de la boîte : $\iint$,
't' une intégrale triple de la hauteur de la boîte : $\iiint$,
'S' le symbole de la sommation, de la dimension de la boîte : Σ,
'P' le symbole du produit de la dimension de la boîte : Π,
'U' le symbole de l'union de la dimension de la boîte : $\cup$,
'I' le symbole de l'intersection de la dimension de la boîte : $\cap$,
'>' une flèche à droite, de la longueur de la boîte : $\rightarrow$,
'<' une flèche à gauche, de la longueur de la boîte : $\leftarrow$,
'^' une flèche vers le haut, de la hauteur de la boîte : $\uparrow$,
'_' une flèche vers le bas, de la hauteur de la boîte : $\downarrow$.

Dans le cas d'une image (type `Picture`), la valeur de la constante est le nom du fichier qui contient l'image, et non l'image numérisée elle-même.

Exemple : les constantes 'Résumé :' et barre de fraction dont on a parlé plus haut se déclarent :

```
CONST
     CsteRésumé = Text 'Résumé :';
     Barre =      Graphics 'h';
```

8.5. Les variables

Les variables permettent d'associer au contenu des boîtes de présentation des valeurs calculées. Une variable porte un nom et a une valeur qui est une chaîne de caractères résultant de la concaténation des valeurs d'une suite de fonctions.

Chaque déclaration de variable est constituée du nom de la variable suivi de ':' et de la suite des fonctions qui produisent sa valeur, simplement séparées par des espaces. Elle se termine par un point-virgule.

```
                'VAR' SuiteVar
SuiteVar =       Variable < Variable > .
Variable =       IdentVar ':' SuiteFonctions ';' .
IdentVar =       NAME .
SuiteFonctions = Fonction < Fonction > .
```

Quatre fonctions sont définies. Les deux premières donnent, sous la forme d'une chaîne de caractères, la date du jour, l'une en anglais (DATE), l'autre en français (FDATE).

Une autre fonction donne simplement la valeur d'une constante de présentation. S'il s'agit d'une constante préalablement déclarée dans la section CONST, il suffit d'indiquer le nom de la constante. Sinon, il faut fournir le type et la valeur de la constante, sous la même forme que dans une déclaration de constante.

La dernière fonction disponible donne, sous la forme d'une chaîne de caractères, la valeur d'un compteur. Cette valeur peut être présentée selon différents styles. Le mot-clé VALUE est suivi entre parenthèses du nom du compteur et du style désiré, ces deux paramètres étant séparés par une virgule. Le style est un mot-clé qui indique si la valeur du compteur doit être présentée en chiffres arabes, en chiffres romains, ou par une lettre.

```
Fonction =       'DATE' / 'FDATE' /
                  IdentConst / TypeConst ValeurConst /
                 'VALUE' '(' IdentCompteur ',' StyleCompteur ')' .
StyleCompteur = 'Arabic' / 'Roman' / 'Alphabetic' .
```

Exemple : pour faire apparaître la date du jour en haut de la première page d'un rapport, une règle CREATE (voir plus loin) associée au type Titre_Rapport engendrera une boîte de présentation dont le contenu (spécifié par la règle Content de cette boîte de présentation) est la variable :

```
VAR
     Date_du_jour : TEXT 'Edité le ' FDATE;
```

Si on a défini les deux compteurs

```
COUNTERS
     CptChapitre : RANK OF Chapitre;
     CptSection :  RANK OF Section;
```

on produira devant chaque titre de section le numéro (en chiffres arabes) de la section précédé du numéro (en chiffres romains) du chapitre, en faisant créer par le titre de section une boîte de présentation dont le contenu est la variable :

```
VAR
     NumSection : VALUE (CptChapitre, Roman) TEXT '.'
                  VALUE (CptSection, Arabic);
```

8.6. Les règles de présentation par défaut

Pour éviter de spécifier, pour chaque type d'élément défini dans le schéma de structure, les valeurs de tous les paramètres de présentation, qui sont nombreux, on définit, après le mot-clé DEFAULT, un ensemble de règles de présentation qui s'appliquent par défaut à toutes les boîtes des éléments définis dans le schéma de structure, à toutes les boîtes de présentation ou de mise en page et à tous les attributs. Seules les règles qui diffèrent de ces règles par défaut sont spécifiées dans la suite du schéma pour chaque type d'élément, d'attribut, ou de boîte de présentation ou de mise en page.

Pour la vue principale, les règles par défaut doivent définir tous les paramètres de présentation, à l'exception des fonctions de présentation (voir plus loin). On peut en plus spécifier des règles pour les autres vues, qui seront alors appliquées comme règles par défaut pour ces vues. Sinon, ce sont les règles par défaut de la vue principale qui s'appliqueront.

Les règles par défaut s'expriment de la même façon que les règles explicites des éléments. Elles sont présentées plus loin.

8.7. Les boîtes de présentation et de mise en page

Dans la présentation on utilise des éléments qui ne font pas partie de la structure logique du document, comme des lignes, des pages, des colonnes (ce sont des boîtes de mise en page) ou encore des filets ou des mots introduisant ce qui suit, comme 'Résumé', 'Annexes', 'Bibliographie', etc... (ce sont des boîtes de présentation). Toutes ces boîtes sont définies, dans le schéma de présentation, dans la section BOXES.

Après le mot-clé BOXES, chaque boîte de présentation ou de mise en page est définie par son nom et une suite de règles de présentation qui indiquent comment elle doit être affichée. Ces règles sont les mêmes que celles qui définissent les boîtes associées aux éléments de la structure logique du document, à l'exception d'une seule, la règle Content qui n'est utilisée que pour les boîtes de présentation, pour en préciser le contenu. Le contenu des boîtes associées aux éléments de la structure du document est défini dans chaque occurrence de document ou d'objet, et n'est donc pas spécifié dans le schéma de présentation, qui s'applique à tous les documents ou objets d'une classe.

Parmi les règles qui définissent une boîte de présentation, certaines peuvent faire référence à une autre boîte de présentation (par exemple des règles de positionnement). Si la boîte référencée est définie après la boîte qui la référence, une instruction FORWARD avec le nom de la boîte référencée doit apparaître avant la référence.

```
                'BOXES' SuiteBoites
SuiteBoites = Boite < Boite > .
Boite =       'FORWARD' IdentBoite ';' /
               IdentBoite ':' SuiteReglesVues .
IdentBoite =  NAME .
```

8.8. La présentation des éléments structurés

Après le mot-clé RULES, le schéma de présentation donne les règles de présentation à appliquer aux éléments dont les types sont définis dans le schéma de structure et qui n'utilisent pas l'ensemble des règles par défaut, définies dans la section DEFAULT du schéma de présentation. En effet, on ne fait figurer dans la section RULES que les types d'éléments auxquels ne s'appliquent pas *toutes* les règles par défaut.

Pour chaque type d'élément, on donne le nom du type, tel qu'il est défini dans le schéma de structure, suivi de ':' et de l'ensemble des règles spécifiques à ce type d'élément, c'est à dire les règles qui diffèrent des règles par défaut.

```
                'RULES' SuitePresent
SuitePresent = Present < Present > .
Present =       IdentType ':' SuiteReglesVues .
```

8.9. La présentation des attributs

Après le mot-clé ATTRIBUTES, tous les attributs dont une ou plusieurs valeurs entraînent une modification de la présentation de l'élément sur lequel ils portent doivent être mentionnés, avec les règles de présentation correspondantes.

Pour chacun de ces attributs, on donne le nom de l'attribut et le nom d'une de ses valeurs, séparés par un signe '=', puis l'ensemble des règles de présentation à appliquer lorsque l'attribut prend cette valeur dans un document. Les règles concernant les attributs sont prioritaires sur les règles par défaut et les règles spécifiques des éléments. Ces règles s'appliquent à tous les éléments terminaux descendant de l'élément portant la valeur correspondante de l'attribut, et uniquement aux éléments terminaux.

```
                'ATTRIBUTES' SuitePresAttr
SuitePresAttr = PresAttr < PresAttr > .
PresAttr =      IdentAttr '=' ValeurAttr ':' SuiteReglesVues .
IdentAttr =     NAME .
ValeurAttr =    NAME .
```

Exemple : si dans le schéma de structure d'un document bilingue on a défini un attribut langue de la façon suivante :

```
ATTR
     Langue = (Français, Anglais);
```

on pourra faire afficher le texte français en caractères romain et le texte anglais en italique par les règles suivantes :

```
ATTRIBUTES
     Langue = Français :
              Highlight: 1;
     Langue = Anglais :
              Highlight: 2;
```

La règle `Highlight` détermine la mise en évidence du texte et notamment le choix des styles de caractères (voir plus loin). Avec ces règles, lorsque l'utilisateur met l'attribut Langue avec la valeur Anglais sur le résumé d'un document, toutes les chaînes de caractères (éléments terminaux) contenues dans le résumé sont affichés en italique.

8.10. Les règles de présentation

Que ce soit pour définir la présentation d'une boîte de présentation ou de mise en page, d'un type d'élément ou d'une valeur d'attribut, dans tous les cas, l'ensemble de règles de présentation est défini de la même façon.

Un tel ensemble de règles est normalement compris entre les mot-clés BEGIN et END, le mot-clé END étant suivi d'un point-virgule. Dans ce bloc, il y a d'abord la suite des règles qui s'appliquent à la vue principale, si les règles par défaut ne conviennent pas. Puis viennent les règles qui s'appliquent spécifiquement aux autres vues, avec une suite de règles pour chaque vue dont les règles par défaut ne sont pas satisfaisantes. Si les règles par défaut des vues non principales conviennent, il n'y a pas de règles spécifiques pour ces vues. Si on ne mentionne qu'une règle, toutes vues confondues, alors les mots-clés BEGIN et END ne figurent pas.

Pour chaque vue on ne précise que les règles qui diffèrent des règles par défaut de la vue, de sorte que, pour certaines vues (ou même toutes), il est possible qu'il n'y ait pas de règle spécifique.

Les règles spécifiques à une vue non principale sont introduites par le mot IN, suivi du nom de la vue. Les règles concernant cette vue suivent, délimitées par les mots-clés BEGIN et END, ou sans ces mots-clés s'il n'y a qu'une règle.

```
SuiteReglesVues = 'BEGIN' < Regle > < ReglesVue > 'END' ';' /
                   ReglesVue / Regle .
ReglesVue =       'IN' IdentVue SuiteRegles .
SuiteRegles =     'BEGIN' Regle < Regle > 'END' ';' / Regle .
```

Exemple : pour que le titre d'un rapport scientifique soit visible dans la vue principale et invisible dans la table des matières et dans la vue des formules, on donne l'ensemble des règles de présentation de la vue principale et une seule règle, qui supprime la visibilité, pour les deux autres vue (la règle `Visibility` est présentée plus loin) :

```
Titre : BEGIN
        Visibility : 1;
        ...     { Autres règles pour la vue principale }
        IN Table_des_matieres
           Visibility : 0;
        IN Formules
           Visibility : 0;
        END;
```

8.11. Une règle de présentation

Une règle de présentation définit soit un paramètre de présentation, soit une fonction de présentation. Les paramètres sont :

- les positions des axes de référence vertical et horizontal de la boîte,
- la hauteur et la largeur de la boîte,
- la position de la boîte par rapport aux autres boîtes,
- l'extensibilité de la boîte,
- le facteur de taille du contenu de la boîte,
- le degré de visibilité de la boîte,
- le degré de mise en évidence du contenu de la boîte,
- les possibilités de modification du contenu de la boîte,
- le retrait de la première ligne de la boîte,
- le contenu de la boîte, pour les boîtes de présentation uniquement.

Les fonctions de présentation sont :

- la création d'une boîte de présentation,
- le mode de mise en page,
- la copie d'une autre boîte.

Pour chaque boîte et dans chaque vue, tous les paramètres de présentation sont définis une fois et une seule, soit explicitement, soit par les règles par défaut. En revanche, les fonctions de présentation ne sont pas obligatoires ou peuvent figurer plusieurs fois pour le même élément. Par exemple, un élément peut créer plusieurs boîtes de présentation. Un autre élément peut n'utiliser aucune fonction de présentation.

Chaque règle définissant un paramètre de présentation commence par un mot-clé suivi de ':'. Le mot-clé indique le paramètre concerné par la règle. Suivant ce mot-clé, le reste de la règle varie. Toutes les règles sont terminées par un point-virgule.

```
Regle =    Regle1 ';' / Regle2 ';' .
Regle1 =  'VertRef' ':' PositionHoriz /
          'HorizRef' ':' PositionVert /
          'Height' ':' Dimension /
          'Width' ':' Dimension /
          'VertPos' ':' PosV /
          'HorizPos' ':' PosH /
          'Expand' ':' Booleen /
          'Break' ':' Booleen /
          'Size'  ':' NombreHerit /
          'Visibility' ':' NombreHerit /
          'HighLight' ':' NombreHerit /
          'Indent' ':' DistanceHerit /
          'Content' ':' VarConst .
Regle2 =   Creation '(' IdentBoite ')' /
           MiseEnPage [ '(' IdentBoite ')' ] /
          'Copy' '(' IdentBoite ')' .
```

8.12. Les axes

La position des axes médians `VMiddle` et `HMiddle` par rapport à leur boîte est toujours calculée automatiquement et n'est donc pas précisée par des règles de présentation. Dans le schéma de présentation, ces axes médians ne sont utilisés que pour positionner leur boîte par rapport à une autre, en spécifiant la distance entre l'axe médian et un axe ou un côté d'une autre boîte (voir les positionnements relatifs).

Les axes de référence d'une boîte sont également utilisés pour positionner la boîte par rapport à une autre, mais, au contraire des axes médians, le schéma de présentation doit expliciter leur position, soit par rapport à un côté ou à l'axe médian de la boîte elle-même, soit par rapport à un côté ou un axe quelconque d'une boîte incluse.

Seules les boîtes des éléments de base ont des axes de référence prédéfinis. Pour les boîtes chaînes de caractères, l'axe de référence horizontal est la ligne de base des caractères (la ligne qui passe par le bas des lettres majuscules, 'Q' excepté) et l'axe de référence vertical est sur le côté gauche du premier caractère de la chaîne.

Les positions des axes de référence d'une boîte sont définies par les règles `VertRef` et `HorizRef` qui précisent la distance entre l'axe de référence et un axe ou un côté parallèle de la même boîte ou d'une boîte incluse. La section suivante indique comment spécifier cette distance.

```
'VertRef' ':' PositionHoriz
'HorizRef' ':' PositionVert
```

Exemple : si dans le schéma de structure des formules la fraction est définie par

```
Fraction = BEGIN
           Numérateur = Expression;
           Dénominateur = Expression;
           END;
```

on peut positionner l'axe de référence horizontal de la fraction, comme dans Edimath, sur le haut du dénominateur par la règle :

```
Fraction :
     BEGIN
     HorizRef : Enclosed Dénominateur . Top;
     ...
     END;
```

Pour placer l'axe de référence horizontal d'un vecteur sur son milieu, comme dans Edimath :

```
Vecteur :
     BEGIN
     HorizRef : * . HMiddle;
     ...
     END;
```

8.13. Les positionnements relatifs

Le positionnement des boîtes utilise les huit axes et côtés, les côtés étant généralement utilisés pour définir des juxtapositions (verticales ou horizontales) de boîtes, les axes médians pour définir des centrages, et les axes de référence des alignements.

Deux règles permettent de positionner une boîte par rapport aux autres. La règle `VertPos` positionne la boîte verticalement, la règle `HorizPos` la positionne horizontalement. Il est possible que la position d'une boîte soit entièrement déterminée par les autres boîtes positionnée par rapport à elle. Dans ce cas la position est implicite, et le mot `nil` permet de ne pas spécifier de règle de positionnement. Dans les autres cas, il faut donner une règle explicite, où on indique l'axe ou le côté qui définira la position de la boîte, puis un signe égal, et la distance entre cet axe ou côté et un axe ou côté parallèle d'une autre boîte, appelée boîte de référence.

```
            'VertPos' ':' PosV
            'HorizPos' ':' PosH
PosV =      'nil' / AxeVert '=' PositionHoriz .
PosH =      'nil' / AxeHoriz '=' PositionVert .
AxeVert =  'Left' / 'VMiddle' / 'VRef' / 'Right' .
AxeHoriz = 'Top' / 'HMiddle' / 'HRef' / 'Bottom' .
```

La boîte de référence est obligatoirement une boîte proche : englobante, englobée ou sœur. Ce peut être soit une boîte de présentation ou de mise en page préalablement définie dans la section `BOXES` du schéma, et créée par une fonction de création ou de mise en page (voir plus loin), soit la boîte associée à un élément structuré.

La position hiérarchique relative de la boîte de référence par rapport à la boîte positionnée est indiquée par un mot-clé : `Enclosing`, `Enclosed`, ou, s'il s'agit d'une boîte de même niveau, `Previous` ou `Next`.

Dans le cas où le mot-clé est ambigu, il est suivi d'un nom de type ou de boîte de présentation ou de mise en page qui lève l'ambiguité ; c'est le cas pour les mots-clés `Enclosed`, `Previous` et `Next`. Si ce nom de type ou de boîte n'est pas indiqué, c'est la première boîte rencontrée qui sera prise comme boîte de référence. Il est aussi possible d'utiliser simplement un nom de type ou de boîte de présentation ou de mise en page, sans le faire précéder d'un mot-clé. Il s'agit alors d'un élément ou d'une boîte de ce type et de même niveau. L'étoile permet de désigner la boîte elle-même (dans ce cas il est évidemment inutile de préciser le type de la boîte de référence). Si le nom de boîte ou de type est précédé du mot `NOT`, il s'agit alors d'une boîte d'un type quelconque, à l'exclusion du type indiqué.

Les mots `Enclosing` et `Enclosed` peuvent être utilisés quel que soit le constructeur du type auquel est associé la règle. S'il s'applique à l'élément représentant tout le document, `Enclosing` désigne la fenêtre ou la page dans laquelle est affichée l'image du document pour la vue concernée par la règle. Le nom de boîte ou de type sans mot-clé est utilisé pour les éléments d'agrégat et désigne un autre élément du même agrégat. On peut aussi l'utiliser pour désigner une boîte de présentation ou de mise en page. Les mots `Previous` et `Next` sont utilisés principalement pour les éléments de liste ou les boîtes de mise en page (ligne précédente, colonne suivante...).

Dans la règle de positionnement, la position hiérarchique relative est suivie, après un point, du nom d'un axe ou d'un côté. Celui-ci indique le côté ou l'axe concerné dans la boîte de référence. Enfin vient la distance, qui indique la distance avec le côté ou l'axe de la boîte de référence. Si cette distance doit être nulle, elle ne figure pas, sinon, c'est un nombre positif ou négatif, le signe n'étant obligatoire que pour les nombres négatifs. Le signe tient compte de l'orientation des axes de coordonnées : de haut en bas pour l'axe vertical et de gauche à droite pour l'axe horizontal. Ainsi une distance négative dans un positionnement vertical indique que le côté ou l'axe spécifié dans la règle est au-dessus du côté ou de l'axe de référence.

Comme le schéma de présentation est indépendant de tout type d'appareil d'affichage, les distances sont exprimées dans une unité "universelle", basée sur la taille des caractères. L'unité de distance est le dixième de la hauteur (le corps) de la police correspondant au facteur de taille de l'élément concerné par la règle. Cette même unité est utilisée pour définir les dimensions des boîtes (hauteur et largeur). Le choix de cette unité assure l'indépendance vis à vis du matériel, mais il permet aussi des changements d'échelle, en ne jouant que sur le paramètre taille des boîtes, ce qui fait changer à la fois, et dans les mêmes proportions, les tailles des caractères et éléments graphiques contenus, les distances entre boîtes et leurs dimensions.

```
PositionVert =  Reference '.' AxeHoriz [ Distance ] .
PositionHoriz = Reference '.' AxeVert [ Distance ] .
Reference =     'Enclosing' /
                'Enclosed' [ NBoiteType ] /
                'Previous' [ NBoiteType ] /
                'Next' [ NBoiteType ] /
                '*' /
                 BoiteType .
BoiteType =      IdentBoite / IdentType .
NBoiteType =     [ 'NOT' ] BoiteType .
Distance =       [ '+' ]  DistancePos / '-' DistanceNeg .
DistancePos =    NUMBER .
```

```
DistanceNeg =   NUMBER .
```

Exemple : si un rapport est défini par le schéma de structure suivant :

```
Rapport = BEGIN
          Titre = Text;
          Résumé = Text;
          ...
          END;
```

on pourra écrire dans le schéma de présentation :

```
Rapport : BEGIN
          VertPos :  Top = Enclosing . Top;
          HorizPos : Left = Enclosing . Left;
          ...
          END;
```

Le rapport est ainsi placé dans le coin supérieur gauche de sa boîte englobante, qui est la fenêtre où le document est édité.

```
Titre :   BEGIN
          VertPos :  Top = Enclosing . Top + 25;
          HorizPos : VMiddle = Enclosing . VMiddle;
          ...
          END;
```

Le haut du titre est à 2,5 lignes (une ligne a la hauteur des caractères du titre) du haut du rapport, donc de la fenêtre d'édition. Le titre est centré horizontalement dans la fenêtre (voir figure 3.3).

```
Résumé :  BEGIN
          VertPos : Top = Titre . Bottom + 15;
          HorizPos : Left = Enclosing . Left + 50;
          ...
          END;
```

Le haut du résumé est placé une ligne et demie en dessous du bas du titre et il est décalé à gauche de 5 caractères par rapport au bord de la fenêtre.

8.14. Les dimensions des boîtes

Les dimensions (hauteur et largeur) de chaque boîte sont définies par les deux règles `Height` et `Width`. Une dimension peut être soit absolue, soit relative. Une dimension absolue impose la hauteur ou la largeur de la boîte. Il s'agit alors d'un simple nombre entier non signé.

Une dimension relative permet de fixer la dimension d'une boîte en fonction de la dimension d'une autre boîte, de la même façon qu'un positionnement relatif place une boîte par rapport à une autre. La boîte qui sert de référence est désignée de la même façon que dans un positionnement relatif. Elle est suivie d'un point et du mot-clé `Height` ou `Width` selon qu'on se réfère à sa hauteur ou à sa largeur. La règle se termine avec le rapport entre la dimension que l'on définit et celle de la boîte de référence. Ce rapport peut être soit un pourcentage soit une différence.

Un pourcentage est indiqué par une étoile (le signe de la multiplication) suivie de la valeur du pourcentage (qui peut être indifféremment supérieure ou inférieure à 100) et par le caractère '%'. Une différence est simplement indiquée par un entier signé.

Un cas particulier de dimensionnement est :

```
Height : Enclosed . Height;
```

ou

```
Width  : Enclosed . Width;
```

qui signifie que la boîte prend une hauteur (ou une largeur) telle qu'elle englobe exactement toutes les boîtes qu'elle contient.

```
                'Height' ':' Dimension
                'Width' ':' Dimension
Dimension =      Reference '.' HautLarg [ Rapport ] /
                 DimensionAbs .
HautLarg =      'Height' / 'Width' .
Rapport =       '*' RapportDim '%' / Distance .
RapportDim =     NUMBER .
DimensionAbs =  NUMBER .
```

Exemple : pour compléter l'exemple précédent, on peut indiquer que le rapport prend la largeur de la fenêtre où on l'édite et la hauteur nécessaire pour son contenu (cette hauteur peut évidemment être plus grande que celle de la fenêtre) :

```
Rapport : BEGIN
          Width : Enclosing . Width;
          Height : Enclosed . Height;
          ...
          END;
```

Le titre occupera 60% de la largeur du rapport (donc de la fenêtre) et sera découpé en lignes sur cette largeur :

```
Titre :   BEGIN
          Width : Enclosing . Width * 60%;
          Height : Enclosed . Height;
          ...
          END;
```

Le résumé occupera toute la largeur de la fenêtre, à l'exception de la marge de 5 caractères qui a été réservée par la règle de positionnement horizontale :

```
Résumé :  BEGIN
          Width : Enclosing . Width - 50;
          Height : Enclosed . Height;
          ...
          END;
```

L'ensemble de ces règles, ajoutées aux règles de positionnement données plus haut, produisent l'agencement des boîtes de la figure 3.3.

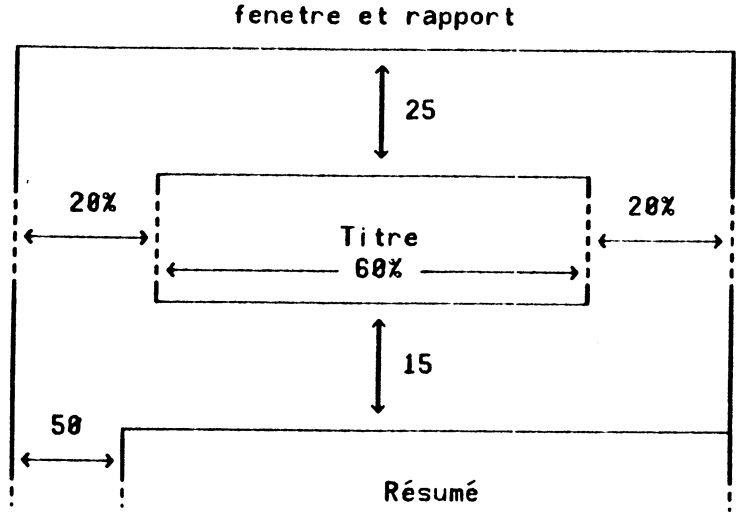


Figure 3.3 : Position et dimension des boîtes.

8.15. L'héritage

Un paramètre peut être défini par référence au même paramètre d'une autre boîte de l'arbre des boîtes. La parenté exprime ce lien structurel. La boîte de référence peut être celle de l'objet immédiatement au-dessus dans la structure (`Enclosing`), immédiatement au-dessous (`Enclosed`) ou immédiatement devant au même niveau (`Previous`).

La valeur du paramètre peut être la même que celle du même paramètre pour la boîte de référence. Cela est indiqué par le signe '='. Mais elle peut aussi être obtenue en ajoutant ou retranchant (selon que le signe est '+' ou '-') un nombre entier à la valeur du

même paramètre de la boîte de référence. Pour maintenir la valeur du paramètre dans certaines limites, il est possible d'imposer une valeur maximum (si le signe est '+') ou minimum (si le signe est '-').

```
Heritage =      Parente  ValeurHerit .
Parente =      'Enclosing' / 'Enclosed' / 'Previous' .
ValeurHerit = '+' EntierPos [ 'Max' maximum ] /
              '-' EntierNeg [ 'Min' minimum ] /
              '=' .
EntierPos =     NUMBER .
EntierNeg =     NUMBER .
maximum =       NUMBER .
minimum =       NUMBER .
```

Les paramètres qui peuvent être obtenus par héritage sont la visibilité, la mise en évidence, la taille, la modifiabilité, l'extensibilité et le retrait.

8.16. Les valeurs numériques

Un certain nombre de paramètres prennent des valeurs numériques. Ces valeurs peuvent être indiquées soit directement par un nombre entier positif, soit par héritage. Ces paramètres sont :

- le degré de visibilité,
- le degré de mise en évidence,
- le facteur de taille.

Notons que si la visibilité est définie par héritage, l'héritage ne peut pas se faire depuis le descendant ni depuis le précédent. La visibilité ne peut s'hériter que de l'ascendant.

```
              'Size'   ':' NombreHerit
              'HighLight' ':' NombreHerit
              'Visibility' ':' NombreHerit
NombreHerit = Entier / Heritage .
Entier =       NUMBER .
```

Le facteur de taille est utilisé d'une part pour déterminer les unités de distance et de dimensions, comme on l'a vu plus haut, et d'autre part pour définir la taille (le corps) des caractères contenus dans la boîte, la largeur de trait des éléments graphiques contenus dans la boîte ainsi que celle des symboles. La valeur 1 représente la plus petite taille, la valeur maximum a été fixée, pour Grif, à 6.

Exemple : si on veut que l'essentiel d'un rapport soit affiché avec des caractères de taille moyenne (par exemple 3), mais que le titre soit affiché avec des caractères plus gros et le résumé avec des caractères plus petits on écrira les règles :

```
Rapport :
     Size : 3;
Titre :
     Size : Enclosing + 2;
Résumé :
     Size : Enclosing - 1;
```

Ainsi il suffira de changer la valeur du paramètre taille du rapport pour changer toutes les tailles de caractères dans le document, tout en conservant les mêmes différences de taille de caractères entre les différents éléments.

Le degré de mise en évidence permet également de jouer sur l'affichage. C'est lui qui détermine le style des caractères contenus dans la boîte, ainsi que leurs attributs d'affichage (style, graisse, italique). Pour le Perq, par exemple, Grif reconnaît trois styles de caractères (Times, Helvetica et Télétype), chacun pouvant être gras ou maigre, droit ou penché.

La correspondance entre les niveaux de mise en évidence et les polices de caractères, avec leurs attributs d'affichage, est définie selon les capacités de l'appareil de restitution. De même, la correspondance entre le facteur de taille et le corps effectif des caractères est définie pour chaque appareil. Ces correspondances sont spécifiées dans des tables que chaque utilisateur peut modifier selon sa convenance et les caractéristiques de l'appareil qu'il utilise.

La visibilité permet de contrôler les éléments qui doivent être affichés et ceux qui ne doivent pas l'être, en fonction du contexte. Il faut noter qu'un élément peut avoir des visibilités différentes dans les différentes vues et que si la visibilité d'un élément est nulle pour une vue, cet élément n'est pas affiché dans cette vue et n'y occupe aucune place (ses dimensions sont nulles).

Exemple : si on ne veut afficher dans la vue Vue_Note que les éléments du type `Note`, on mettra dans les règles par défaut :

```
DEFAULT
     IN Vue_Note Visibility:0;
```

ce qui assurera que tous les éléments seront invisible dans la vue Vue_Note. Mais on associera au type `Note` la règle spécifique :

```
Note :
     IN Vue_Note Visibility:1;
```

La visibilité peut prendre d'autres valeurs que 0 ou 1. Si des valeurs supérieures à 1 sont utilisées, elles permettent à l'utilisateur, lors de l'édition, de spécifier le degré de visibilité qui l'intéresse, et ainsi de ne voir que les boîtes dont le degré de visibilité est supérieur à un certain seuil. L'utilisateur a alors le contrôle de la granularité des images qui sont affichées.

8.17. Les Booléens

D'autres paramètres de présentation sont de simple booléens. Il s'agit de l'extensibilité de la boîte et de la sécabilité de la boîte. La valeur de ces paramètres est donnée directement par un mot-clé `Yes` ou `No`.

```
            'Expand' ':' Booleen
            'Break' ':' Booleen
Booleen = 'Yes' / 'No' .
```

L'extensibilité est utilisée pour indiquer si une boîte de mise en page, comme une boîte ligne, doit être étirée de façon à occuper toute la largeur de la boîte qui l'englobe. C'est ce qui permet de contrôler la justification des lignes.

La règle `Break` permet d'indiquer si la boîte des éléments d'un type donné peut être coupée lors de la construction pages. Une boîte sécable peut être coupée, une partie en bas d'une page (ou d'une colonne), et l'autre partie en haut de la page (ou de la colonne) suivante.

8.18. Le retrait

Le retrait n'est utilisée que pour les éléments qui sont mis en ligne. Il détermine le décalage horizontal de la première ligne de l'élément par rapport aux autres lignes du même élément. Le retrait peut être indiquée par une valeur immédiate ou par héritage. La valeur immédiate est un nombre entier négatif (décalage à gauche), nul (pas de décalage) ou positif (décalage à droite, le signe est facultatif). L'unité de distance est toujours la même : le dixième de la hauteur des caractères avec le facteur de taille de l'élément auquel se rapporte la règle de retrait.

S'il n'est appliquée qu'aux éléments mis en lignes (voir l'instruction `Line`), le retrait peut néanmoins être défini pour n'importe quelle boîte, et transmis par héritage jusqu'aux éléments mis en ligne.

```
            'Indent' ':' DistanceHerit
DistanceHerit = Distance / Heritage .
```

Exemple : les règles suivantes permettent d'obtenir un retrait nul pour les paragraphes du résumé et un retrait de cinq caractères pour les paragraphes du corps du document, tout en utilisant le même type d'élément paragraphe dans le résumé et dans le corps.

```
Résumé :
     Indent : 0;
Corps :
     Indent : 50;
Paragraphe :
     BEGIN
     Indent : Enclosing =;
     Line (Ligne_justifiée);
     END;
```

8.19. Le contenu

Ce paramètre s'applique uniquement aux boîtes de présentation. Il indique le contenu imposé à une boîte. Ce contenu est soit la valeur d'une variable, soit la valeur d'une constante.

S'il s'agit d'une constante, on peut, comme dans une déclaration de variable, indiquer soit le nom d'une constante déclarée dans la section CONST, soit directement le type et la valeur du contenu de la boîte.

De même, s'il s'agit d'une variable, on peut soit donner le nom de la variable si elle a été déclarée dans la section VAR, soit définir la variable entre parenthèses. Le contenu de la parenthèse est alors constitué de la même façon qu'une déclaration de variable.

```
          'Content' ':' VarConst VarConst = IdentConst / TypeConst
ValeurConst /
           IdentVar / '(' SuiteFonctions ')' .
```

Exemple : le contenu de la boîte de présentation créée pour faire apparaître le numéro de chapitre et le numéro de section devant chaque titre de section peut être défini par

```
BOXES
     BoiteNumSection :
          BEGIN
          Content : NumSection;
```

```
        ...
        END;
```

si la variable NumSection a été définie dans la section des définitions de variables du schéma de présentation. Sinon la règle content s'écrira

```
BOXES
    BoiteNumSection :
        BEGIN
        Content : (VALUE (CptChapitre, Roman) TEXT '.'
                VALUE (CptSection, Arabic));
        ...
        END;
```

8.20. La création de boîtes de présentation

Une règle de création indique qu'une boîte de présentation doit être créée lorsqu'apparaît dans le document un élément du type auquel se rapporte la règle. Cette création peut être systématique ou conditionnelle. La condition porte sur le fait que l'élément a ou n'a pas de prédécesseur ou de successeur à son niveau dans la structure logique du document.

Une création conditionnelle est indiquée par le mot-clé IF. Ce mot est éventuellement suivi du mot-clé NOT et, obligatoirement, d'un des deux mots-clés FIRST ou LAST, avec la signification suivante :

IF FIRST : la boîte est créée seulement si l'élément est le premier.
IF NOT FIRST : la boîte est créée, sauf si l'élément est le premier.
IF LAST : la boîte est créée seulement si l'élément est le dernier.
IF NOT LAST : la boîte est créée, sauf si l'élément est le dernier.

Ces conditions représentent le minimum qu'il faut offrir. Elles se révèlent suffisantes dans de nombreux cas, mais dès qu'il s'agit d'une mise en page évoluée, d'autres conditions doivent être proposées, comme notamment la parité : les pages paires et impaires ont souvent un aspect différent. C'est un des points qui justifierait une extension du langage.

Après la condition, un mot-clé indique la position que prendra dans la structure la boîte à créer, par rapport à la boîte de l'élément concerné par la règle :

Create indique que la boîte doit être créée comme premier élément du niveau inférieur, avant ceux qui existent déjà ;
CreateBefore indique que la boîte doit être créée au même niveau que la boîte créatrice et devant elle;

`CreateAfter` indique que la boîte doit être créée au même niveau et après la boîte créatrice.

Le type de la boîte de présentation qui doit être créée est indiqué entre parenthèses après ce mot-clé. Les règles de création de boîtes ne peuvent pas être utilisées dans la définition d'une boîte de présentation : une telle boîte ne peut pas en créer une autre. Elles ne peuvent pas non plus figurer dans les règles de présentation par défaut. D'autre part, les boîtes créées doivent nécessairement posséder une règle `Contenu`.

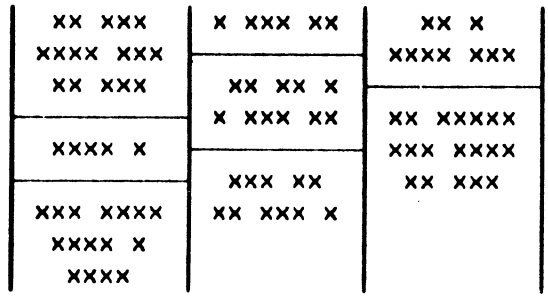
Les règles de création ne peuvent apparaître que dans un bloc de règles qui se rapporte à la vue principale ; la création est provoquée par un élément du document et pour toutes les vues. Cependant, pour chaque vue, la boîte de présentation n'est créée que si l'élément créateur a lui-même une boîte dans la vue. De plus, dans les règles définissant la présentation de la boîte créée, on peut jouer sur la visibilité dans chaque vue pour contrôler la création de la boîte de présentation vue par vue.

```
            Creation '(' IdentBoite ')'
Creation =  [ 'IF' Condition ] Cree.
Condition = [ 'NOT' ] FirstLast .
FirstLast = 'First' / 'Last' .
Cree =      'Create' / 'CreateBefore' / 'CreateAfter' .
```

Exemple : On définit un type d'objet, appelé Table, qui est constitué d'une suite de colonnes de même largeur, fixe, les colonnes étant séparées les unes des autres par des filets verticaux. Il y a un filet à gauche de la première colonne et un à droite de la dernière. Chaque colonne a un nombre variable de cellules, positionnées l'une en dessous de l'autre et séparées par des filets horizontaux. Il n'y a pas de filet au-dessus le la première cellule ni au-dessous de la dernière. Le texte contenu dans chaque cellule est découpé en lignes et ces lignes sont centrées horizontalement dans la cellule. La structure logique de cet objet est définie par

```
Table =   LIST OF (Colonne);
Colonne = LIST OF (Cellule = Text);
```

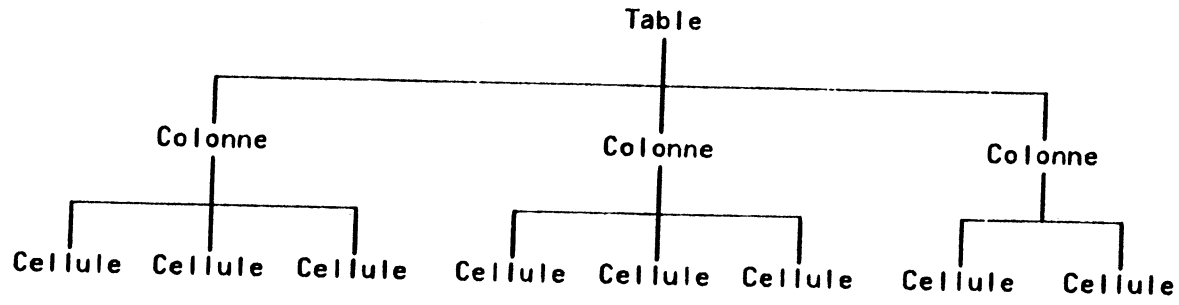
Sa présentation doit ressembler au dessin suivant :



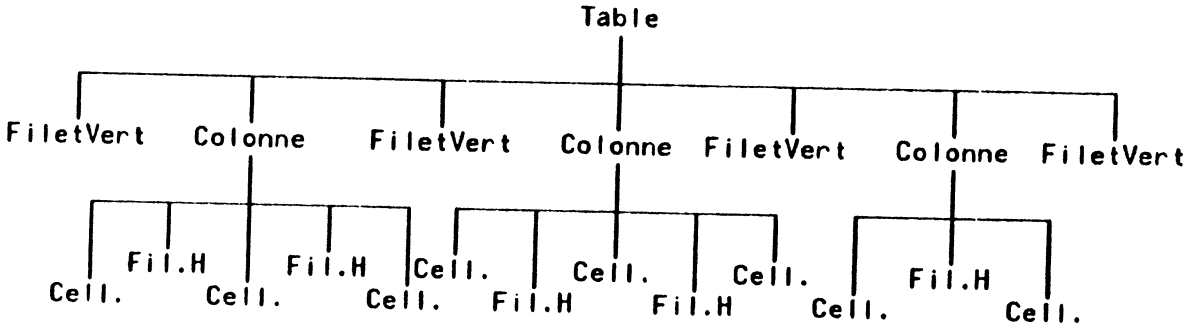
Elle est définie par l'extrait de schéma de présentation suivant :

```
BOXES
     FiletVert : BEGIN
                 Width : 1;
                 Height : Enclosing . Height;
                 VertPos : Top = Enclosing . Top;
                 HorizPos : Left = Previous . Right;
                 Content : Graphics 'v';
                 END;

     FiletHoriz: BEGIN
                 Width : Enclosing . Width;
                 Height : 10;
                 VertPos : Top = Previous . Bottom;
                 HorizPos : Left = Enclosing . Left;
                 Content : Graphics 'h';
                 END;

     Ligne_centree :
                 HorizPos : VMiddle = Enclosing . VMiddle;
RULES
     Colonne : BEGIN
               CreateBefore (FiletVert);
               IF LAST CreateAfter (FiletVert);
               Width : 100;
               Height : Enclosed . Height;
               VertPos : Top = Enclosing . Top;
               HorizPos : Left = Previous . Right;
               END;

     Cellule : BEGIN
               IF NOT FIRST CreateBefore (FiletHoriz);
               Width : Enclosing . Width;
               Height : Enclosed . Height;
               VertPos : Top = Previous . Bottom;
               HorizPos : Left = Enclosing . Left;
               Line (Ligne_centree);
               END;
```

On remarque que la règle de positionnement horizontal du premier filet vertical ne peut pas être appliquée, puisqu'il n'y a pas de boîte précédente. Dans ce cas, la boîte est simplement positionnée sur le côté gauche de sa boîte englobante. La figure 3.4 montre la structure logique du document et la structure de ses boîtes, avec les créations.



la structure logique



l'arbre des boîtes

Figure 3.4 : Création de boîtes de présentation.

8.21. La mise en page

Trois fonctions de mise en page sont définies : `Line`, `Column` et `Page`.

La règle `Line` indique que le contenu de la boîte doit être mis en lignes : les boîtes incluses dans la boîte à laquelle se rapporte la règle sont affichées les unes à la suite des autres, de gauche à droite, leurs axes de référence horizontaux étant alignés, de façon à former des lignes. Lorsque la ligne construite devient trop longue (plus longue que la largeur de la boîte portant la règle `Line`), une nouvelle ligne est créée pour les boîtes incluses suivantes. Une nouvelle ligne peut également être créée lorsqu'une boîte incluse contient une chaîne de caractères trop longue. Dans ce cas la boîte chaîne est coupée sur plusieurs lignes. Si les boîtes incluses comportent des règles de positionnement relatif, ces règles sont ignorées, les boîtes étant placées selon la règle de mise en lignes.

Si le mot-clé `Line` est suivi, entre parenthèses, d'un nom de boîte, les règles de cette boîte, dite boîte ligne, seront utilisées pour la constitution des lignes. C'est avec ces règles qu'on détermine l'interligne, la justification, ou le centrage des lignes.

L'interligne est défini en positionnant l'axe de référence horizontal d'une boîte ligne par rapport à celui de la boîte ligne précédente. Par exemple, la règle

```
Boite_Ligne :
   VertPos : HRef = Previous . HRef + 15;
```

définit un interligne de 1,5.

La justification s'obtient en rendant les boîtes lignes extensibles horizontalement, de sorte qu'elles prennent la largeur de la boîte qui les englobe (la boîte qui porte la règle `Line`) :

```
Boite_Ligne_Justifiée :
   Expand : Yes;
```

L'alignement à droite s'obtient en positionnant le côté droit des boîtes lignes sur le côté droit de la boîte englobante :

```
Boite_Ligne_à_Droite :
   HorizPos : Right = Enclosing . Right;
```

L'alignement à gauche s'obtient de façon analogue :

```
Boite_Ligne_à_Gauche :
   HorizPos : Left = Enclosing . Left;
```

Enfin, le centrage des lignes produites par le processus de mise en lignes s'obtient en positionnant l'axe médian vertical des boîtes lignes sur celui de leur boîte englobante :

```
Boite_Ligne_Centrée :
   HorizPos : VMiddle = Enclosing . VMiddle;
```

Si la règle `Line` n'indique pas de boîte, les boîtes englobées seront bien alignées, mais une seule ligne sera créée, qui sera la boîte à laquelle est associée la règle `Line` et cette boîte aura les dimensions minimales pour encadrer l'ensemble des boîtes englobées.

Les règles `Column` et `Page`, procèdent de même : elles construisent des boîtes colonnes ou page selon les règles attachées à ces boîtes de mise en page.

```
                    MiseEnPage [ '(' IdentBoite ')' ]
MiseEnPage = 'Line' / 'Column' / 'Page' .
```

Exemple : l'exemple précédent montre comment le texte des cellules est découpé en lignes qui sont centrées dans les cellules.

8.22. Les copies de boîtes

La règle Copie n'est utilisable que pour un élément qui est défini comme une référence dans le schéma de structure. Cette règle indique la boîte qui doit être produite lorsque cette référence apparaît dans la structure. La boîte produite est une copie (même contenu, mais présentation éventuellement différente) de la boîte du type indiqué en paramètre de la règle et qui est dans l'élément référencé.

```
'Copy' '(' IdentBoite ')' .
```

Exemple : si on a dans le schéma de structure les définitions suivantes :

```
Corps = LIST OF (Chapitre =
                    BEGIN
                    Titre_Chapitre = Text;
                    Corps_Chapitre = Suite_Sections;
                    END);
Ref_Chapitre = REFERENCE (Chapitre);
```

on peut spécifier les règles de présentation suivantes, parmi les autres règles du schéma de présentation :

```
COUNTERS
   Cpt_Chapitre : RANK OF Chapitre;
BOXES
   Numéro_Chapitre :
      Content : (VALUE (Cpt_Chapitre, Roman));
RULES
   Chapitre :
      Create (Numéro_Chapitre);
   Ref_Chapitre :
      Copy (Numéro_Chapitre);
```

qui ont pour effet de faire apparaître, à la place de la référence à un chapitre, le numéro, en chiffre romain, du chapitre référencé. Mais si on souhaite plutôt faire apparaître le titre du chapitre à la place de la référence, la règle Copy doit être écrite :

```
Copy (Titre_Chapitre);
```

9. CONCLUSION

Nous avons développé dans ce chapitre un modèle à la fois souple et complet qui permet une représentation structurée de documents très divers ainsi que des objets contenus dans les documents. Nous avons aussi proposé un modèle de présentation permettant d'associer à chaque modèle structurel de document ou d'objet une ou plusieurs présentations génériques, définissant l'image des documents sur des supports de visualisation quelconques. Ces modèles peuvent être exprimés dans des langages adaptés.

Il reste maintenant à mettre en œuvre les modèles de structure et de présentation dans un système de manipulation de document et, d'abord, à réaliser les compilateurs qui acceptent les langages de spécification de structure (S) et de présentation (P). C'est l'objet du chapitre suivant.

CHAPITRE 4

L'EDITEUR DE DOCUMENTS GRIF

1. PRESENTATION

Ce chapitre présente l'éditeur Grif [Quint 86a, Quint 86c], qui a été réalisé pour mettre en œuvre, dans un environnement interactif, le modèle de document développé dans le chapitre précédent. Ce même modèle, élaboré au sein du projet Tigre [Bogo], guide également la réalisation du système de gestion de bases de données Tigre [Velez, Lopez], non décrit ici.

Après une discussion des principes qui ont présidé à la conception de l'éditeur, nous présentons son architecture générale, puis les compilateurs des langages S et P, enfin l'éditeur lui-même, en séparant les problèmes de manipulation de structure des problèmes d'affichage et de gestion de l'interface utilisateur.

Cette description de l'éditeur met l'accent sur les manipulations de la structure logique des documents et sur la construction et la mise à jour des images affichées sur l'écran. L'interface utilisateur n'est pas présentée de façon détaillée dans la mesure où elle n'est pas figée. Un des choix de réalisation de Grif, on le verra, est précisément de permettre l'adaptation de l'interface à l'utilisateur. L'annexe 5 donne un certain nombre de détails techniques concernant la réalisation de Grif.

2. LES PRINCIPES D'EDITION

On oppose généralement deux approches dans la conception des systèmes de production de documents (voir notamment l'introduction de [Lamport]). Il y a d'une part l'approche WYSIWYG, qui apporte un grand confort d'utilisation, mais limite les traitements possibles et la qualité des documents imprimés. Il y a d'autre part l'approche formateur, qui conduit à des systèmes d'un emploi moins aisé, mais plus puissants et plus riches en possibilités typographiques. Avec Grif nous avons essayé d'échapper à ce dilemme et de reprendre le meilleur de chacune des deux approches, en les considérant comme complémentaires plutôt qu'antagonistes.

2.1. L'image du document

Ce qui nous paraît le plus intéressant dans l'approche WYSIWYG ce n'est pas le fait d'obtenir sur le papier une image identique à ce que le système affiche sur l'écran pendant l'édition du document. En effet, il est dommage de se limiter à la qualité d'une image d'écran, puisqu'à l'impression des algorithmes sophistiqués peuvent être mis en œuvre pour la construction des lignes [Plass], la coupure des mots [Liang], ou la constitution des pages, alors que l'aspect "temps réel" de l'édition interdit de les utiliser pour élaborer l'image affichée sur l'écran. D'autres traitements, comme la constitution d'index ou de tables de références, ne peuvent pas non plus être effectués dynamiquement au cours de l'édition et l'image d'écran ne peut donc pas présenter ces index et ces tables. La forme finale sur papier devant être identique à celle affichée sur l'écran, les documents traités de cette façon n'ont ni tables de références ni index. De plus notre modèle prévoit plusieurs présentations différentes pour la même classe, donc pour le même document, sans intervention "manuelle" de l'utilisateur, ce qui s'oppose à l'approche WYSIWYG.

D'ailleurs, avec la plupart des systèmes WYSIWYG, s'il est nécessaire de pouvoir contrôler à l'écran la mise en page des documents, c'est précisément parce que le système n'effectue pas un formatage sans reproche : les défauts sont visibles et l'utilisateur peut les corriger "à la main". Mais si l'utilisateur sait que, lors de l'impression, il peut compter sur une mise en page parfaite, ou tout au moins meilleure que celle qu'il pourrait réaliser lui-même, l'intérêt d'un contrôle du formatage à l'écran s'en trouve réduit. S'il sait que le système effectue des coupures de mots correctes (voir notamment [Liang] et [Désarménien]), qu'il ne produit pas de ligne isolée en haut ou bas de page, que les chapitres débutent toujours sur des pages impaires, etc, il n'a pas à s'en préoccuper lors de l'édition.

L'intérêt des systèmes WYSIWYG réside plutôt dans l'interactivité, qui rend plus facile l'accès au système et permet à l'utilisateur de vérifier en permanence ce qu'il fait. Concernant ce dernier point, l'important est que l'utilisateur agit en temps réel sur une forme directement lisible du document. Il voit immédiatement comment le système interprète chaque commande. C'est ce point que nous retenons d'abord pour notre éditeur : même si l'image affichée n'est pas celle (ni la seule) qu'on peut obtenir sur le papier, elle offre néanmoins à l'utilisateur le moyen de vérifier immédiatement que les commandes qu'il entre sont correctes, qu'elles expriment effectivement ce qu'il veut faire, et que le système les interprète bien comme prévu.

Une notion importante dans les systèmes WYSIWYG est la notion de page. Elle ne nous semble pas fondamentale pour Grif, puisqu'elle n'apporte pas d'information supplémentaire à l'utilisateur pour contrôler son travail et qu'elle peut très bien être introduite uniquement lors de l'impression du document. C'est le support papier qui impose le découpage en pages ; l'écran, lui, permet tout à fait de présenter un document comme un

rouleau continu ou un ensemble de rouleaux, ce que fait Grif. L'absence de pages lors de l'édition simplifie notablement le formatage du document sur l'écran et améliore donc les performances de l'éditeur.

Un autre point fort des systèmes WYSIWYG est la qualité de l'image présentée à l'utilisateur. Prétendant imiter la typographie, ces systèmes construisent, pendant la phase d'édition, des pages beaucoup plus lisibles que ce qu'on peut voir sur l'écran où l'on édite un texte destiné à un formateur, constellé de caractères et de commandes bizarres qui rendent la lecture chaotique. Or le confort de lecture détermine très directement la facilité de correction. Avant de pouvoir opérer toute modification, il faut localiser, dans le document, la partie à modifier ; et cette localisation est d'autant plus rapide que le document peut être parcouru plus aisément.

Dans Grif nous avons donc cherché à afficher sur l'écran une image de qualité comparable à celle que produisent des systèmes comme le Star [Smith] ou Interleaf [Morris 85], non pas dans le but de donner l'image précise de ce qui sera imprimé, mais plutôt avec le souci de rendre la lecture plus aisée et d'offrir un contrôle efficace sur les traitements effectués par l'éditeur.

2.2. Les commandes et les traitements

Un autre aspect de l'interactivité est le mode d'entrée des commandes. Alors que les formateurs imposent à l'utilisateur la connaissance d'une syntaxe rigide et d'un large vocabulaire, les systèmes interactifs sont d'un abord plus aisé, avec leur souris, leurs menus, leurs touches de fonction et autres messages d'aide. Grif propose ce dernier type d'interface utilisateur. Mais nous n'avons pas voulu fixer de façon définitive le mode d'entrée de chaque commande. Vaut-il mieux un menu, une touche de fonction ou une commande tapée au clavier ? Faut-il demander les commandes avant leurs paramètres ou l'inverse ? Nous n'avons pas souhaité prendre ce genre de décision d'une façon générale, pour toutes les commandes ou tous les utilisateurs. Aussi avons-nous choisi de laisser à chaque utilisateur le soin de faire ce type de choix, selon ses habitudes, ses goûts ou son degré de connaissance du système.

Les commandes à la disposition de l'utilisateur reflètent les traitements réalisables par le système. Grâce au modèle retenu, Grif peut effectuer des traitements puissants sur les documents, du même ordre que ceux qui sont réalisés par les formateurs qui considèrent la structure des documents, comme Scribe, LaTeX ou Mint. C'est aussi cet aspect des choses (des traitements puissants) qui rend les commandes d'un formateur plus difficiles à manipuler que celles d'un éditeur interactif. Comme, sur ce point, Grif se rapproche des formateurs, une grande attention a été apportée à la définition des commandes, et en

particulier au niveau de leur généricité : chaque fois que c'était possible, nous avons cherché à utiliser la même commande pour effectuer des traitements analogues à différents niveaux.

Les traitements de Grif sont en grande partie fondés sur la structure des documents ; il était donc tentant de faire apparaître explicitement cette structure dans les commandes, en particulier pour la sélection ou les déplacements. Mais l'utilisateur n'est pas nécessairement prêt à entrer dans les détails du modèle et il n'est pas censé être habitué aux manipulations d'arbres. Aussi avons-nous cherché à lui présenter cette structure non pas comme une arborescence abstraite, comme le font souvent les éditeurs syntaxiques de programmes, mais plutôt à travers son image sur l'écran. Les relations structurelles sont en effet traduites en relations géométriques sur l'écran ; elles sont directement visibles et peuvent donc être manipulées graphiquement.

Il faut ici faire une distinction entre deux catégories d'utilisateurs. Il y a les administrateurs du système, qui connaissent bien le méta-modèle et les langages S et P, et qui sont capables de créer de nouvelles classes de documents ou d'objets et de nouveaux schémas de présentation. On peut comparer ce rôle à celui de l'administrateur de la base de données de Scribe ou de Mint. Et puis il y a les utilisateurs finaux, qui créent et éditent des documents, en utilisant les schémas que les précédents mettent à leur disposition. Si les administrateurs du système sont bien des programmeurs, les utilisateurs finaux, eux, ne doivent pas avoir de connaissances particulières de la programmation pour pouvoir travailler. C'est notamment pour cette raison qu'on évite de leur demander de penser leur document en termes d'arbre, et qu'on cherche une interface d'utilisation plus naturelle.

Un autre écueil de l'édition structurée vient de la frontière, souvent très nette, qui sépare les manipulations de structure des manipulations d'éléments non structurés constituant les atomes de la structure. Certains systèmes, comme Mentor-Rapport [Mélèse], imposent même à l'utilisateur un changement complet d'environnement. Nous avons au contraire cherché à estomper autant que possible cette frontière, et à offrir des commandes génériques qui réalisent le même type d'opération aussi bien sur la structure que sur le contenu des atomes (sélection, insertion, suppression,...).

2.3. L'impression des documents

Dès la définition du modèle de document, et donc *a fortiori* lors de la conception de l'éditeur, nous avons rejeté l'idée d'une présentation unique pour chaque document. L'éditeur doit cependant créer une image des documents sur l'écran. Il doit aussi permettre de produire une (ou plusieurs) image(s) des documents sur papier. Comme l'image de l'écran n'est pas l'image finale, l'éditeur peut se limiter à un compromis entre la rapidité de

calcul de cette image et sa qualité graphique. Ainsi, par exemple, il peut couper le texte en lignes, justifier ou centrer ces lignes, mais il ne coupe pas les mots en bout de ligne, pas plus qu'il n'augmente les espaces en fin de phrase ou autour des opérateurs dans les expressions mathématiques.

Pour l'impression, au contraire, aucun compromis n'est possible si on veut obtenir la meilleure qualité : il faut respecter toutes les règles de la typographie. Mais il n'est pas nécessaire de réaliser la composition du document à imprimer au moment même de son édition. Ce traitement, lourd si on recherche une bonne qualité, peut très bien être réalisé indépendamment, pourvu que le document comporte toutes les informations nécessaires. Le modèle de document utilisé dans Grif est précisément fondé sur cette idée que la présentation, moyennant quelques règles, peut s'effectuer à partir de la structure. Si l'éditeur produit une représentation structurée du document, la composition peut être réalisée indépendamment de l'édition.

Ce principe étant posé, il reste à effectuer la composition. Il existe des formateurs dont c'est précisément la tâche. Seulement, on ne peut pas soumettre directement à un formateur un document produit par Grif. Plutôt que de développer un formateur acceptant en entrée le format de Grif ou d'adapter à ce format un formateur particulier, nous avons choisi une solution plus souple, qui consiste à traduire un document venant de Grif dans le langage d'entrée d'un formateur. De façon à ouvrir le choix des formateurs, cette traduction est paramétrée : un langage spécifique, le langage F, permet de définir les règles de traduction pour chaque couple formateur-classe de document. Ce travail présenté dans [Quint 86b] est mené par H. Bedor et doit faire l'objet de sa thèse.

Une autre possibilité pour la composition des documents serait d'utiliser les algorithmes de présentation mis en œuvre dans l'éditeur Grif pour construire, à partir de la structure et d'un schéma de présentation, une image du document, exactement comme le fait l'éditeur lorsqu'il affiche un document sur l'écran. Mais on a vu que cet affichage se faisait avec une qualité qui paraitrait médiocre sur papier. Il faudrait donc intégrer dans le processus de présentation les bons algorithmes de coupure de mots, de coupure de lignes, de formation de paragraphes, de calcul de pages, de crénage, etc... L'utilisation des formateurs intégrant déjà ces algorithmes nous a semblé préférable. Néanmoins, le formateur Sroff a été développé spécifiquement pour Grif, car il n'existait pas de formateur disponible pour notre environnement logiciel et matériel offrant les qualités typographiques nécessaires.

3. LES FONCTIONALITES DE L'EDITEUR

Partant de ces principes et du modèle de document, nous avons défini les principales fonctionalités de l'éditeur, telles que l'utilisateur peut les voir.

3.1. Les vues

Grif autorise la manipulation simultanée de plusieurs documents différents, appartenant ou non à la même classe. Cela permet notamment de copier ou de déplacer une partie d'un document vers un autre document. Pour chaque document en cours de traitement, Grif accepte de travailler sur plusieurs vues, chacune présentant

- soit tous les éléments du document (c'est généralement le cas de la vue principale),
- soit seulement certains types d'éléments (pour montrer d'une façon plus synthétique ce qui se trouve également dans la vue principale),
- soit tous les éléments associés d'un type donné (pour complèter la vue principale).

Chaque vue est affichée sur l'écran dans une fenêtre que l'utilisateur peut contrôler. Lors de la création d'une vue, il choisit la position et la dimension de la fenêtre correspondante. A tout moment, il peut déplacer ou changer la taille d'une fenêtre affichée. Si les règles de présentation indiquent que l'affichage de certains éléments dépend des dimensions de la fenêtre (centrages, longueurs des lignes notamment), ces éléments sont remis en forme lors des modifications de la fenêtre.

Chaque document en cours d'édition est présenté à travers une vue au moins. Dès que le document est chargé dans l'éditeur sa vue principale est automatiquement affichée. Ensuite, d'autres vues peuvent être créées ou détruites au gré de l'utilisateur, tout en laissant le document éditable. Lorsqu'il ne reste qu'une vue d'un document à l'écran et que celle-ci est supprimée, le document n'est plus accessible à l'utilisateur, et l'éditeur le décharge.

La vue principale ne se distingue des autres vues que lors du chargement du document, puisqu'elle est automatiquement créée à ce moment. Elle peut ensuite être supprimée et recréée comme n'importe quelle autre vue ; elle n'est nécessaire pour aucun traitement et toutes les opérations d'édition peuvent s'effectuer indistinctement sur toutes les vues. En particulier, il est possible de modifier un document, d'opérer une sélection ou de se déplacer dans le document à travers n'importe quelle vue. Il est même possible de commencer une opération sur une vue et de la poursuivre sur une autre. Les vues d'un même document ont évidemment des relations entre elles, et lorsqu'une opération est effectuée à travers une vue, son résultat est reporté automatiquement sur toutes les vues du

même document où ce résultat est visible. L'utilisateur peut par exemple entrer les titres de section dans la vue de la table des matières et entrer leur contenu dans la vue principale, la vue principale affichant néanmoins tous les titres. Une autre utilisation de cette cohérence entre les vues consiste à se déplacer dans le document à travers la vue de la table des matières et, lorsque la partie cherchée est atteinte, à la modifier sur la vue principale.

3.2. L'édition

Grif tire évidemment parti de la structure des documents qu'il manipule. L'édition est donc largement guidée par les structures génériques, et notamment pour ce qui concerne les opérations de création, sélection ou déplacement.

Mais il existe aussi dans les documents des parties non structurées, qui sont les feuilles de la structure. Pour ces parties, et particulièrement les parties textuelles, nous avons retenu le même type de manipulation que dans les systèmes WYSIWYG. Il n'y a pas de commande explicite d'insertion de texte ; les caractères frappés au clavier viennent s'insérer à la position courante du curseur, qui peut être déplacé à tout moment.

L'effacement peut se faire caractère par caractère, à la position courante du curseur, grâce à une touche de fonction. Cette façon de faire convient bien aux corrections d'erreurs de frappe, mais, pour les modifications plus importantes, une autre méthode est utilisable, qui consiste à sélectionner d'abord la partie à effacer, puis à entrer une commande qui efface effectivement la partie sélectionnée.

De façon à conserver une interface utilisateur homogène, nous avons reporté au niveau de la structure le même type de traitement. L'insertion de nouveaux éléments structurels se fait ainsi à l'endroit du curseur ou de la sélection courante. La suppression d'éléments de structure se fait par sélection puis commande de suppression. La sélection dans la structure peut s'effectuer simplement en désignant sur l'écran, avec la souris, le début et la fin de la zone à sélectionner. Dans le cas où on souhaite supprimer en une même commande une partie de texte, quelques éléments de structure et enfin une dernière partie de texte, l'éditeur prend en charge lui-même la répartition des traitements entre la structure et le contenu, sans imposer à l'utilisateur des changements de contexte pour traiter les différents types d'éléments, selon qu'ils sont structurés ou non.

Cette homogénéité de l'interface est obtenue par un petit nombre de commandes génériques qui s'appliquent aussi bien aux parties non structurées (le contenu des feuilles) qu'à la structure elle-même. Grif offre quatre commandes de ce type, qui sont inspirées de l'environnement Smalltalk [Goldberg] :

- Insérer
- Couper
- Coller
- Copier

Ajoutées au modèle uniforme qui décrit de la même façon le document lui-même et les objets qu'il contient, ces commandes génériques, qui agissent sur tous les types de structure (document, objets) et sur le contenu, assurent une grande homogénéité des traitements et de l'interface utilisateur. On est ainsi loin des systèmes qui traitent les documents complexes en fournissant à l'utilisateur une collection d'éditeurs spécialisés, chacun ayant ses particularités.

4. L'ARCHITECTURE GENERALE DE GRIF

L'architecture générale de Grif est résumée dans la figure 4.1. Le comportement du système est défini par trois ensembles de règles, fournis de façon externe à Grif. Au centre se trouvent les *schémas de structure* qui définissent les règles de structuration des documents et des objets. Ces schémas sont écrits par un administrateur du système, dans le langage S. Pour l'édition Grif utilise aussi des *schémas de présentation*, un pour chaque classe de document ou d'objet qu'il traite. Ces schémas de présentation sont également écrits par un administrateur du système, dans le langage P. Enfin, pour traduire les documents destinés à un formateur, Grif utilise des *schémas de traduction* exprimées dans le langage F.

Pour chacun de ces trois langages il existe un **compilateur** qui produit des tables. Ce sont ces tables qui sont ensuite directement utilisées par Grif pour ses traitements. Les compilateurs ne sont normalement utilisés que par un administrateur du système, et seulement lors de la création ou de la modification des schémas. Une fois qu'un schéma est au point, seules les tables produites par les compilateurs sont utilisées.

L'éditeur Grif est formé de deux composants principaux, l'Editeur et le Médiateur, qui prennent chacun en charge un aspect de l'édition. L'**Editeur** effectue essentiellement les manipulations de structures et utilise les *tables de structure* et les *tables de présentation* produites par les compilateurs S et P. Il travaille sur la représentation logique des documents, les *arbres abstraits*. Il prépare l'affichage en décrivant des *images abstraites*. Le **Médiateur** gère l'interface utilisateur et l'affichage. Il construit notamment les *images concrètes* présentées sur l'écran. L'Editeur produit un fichier pour chaque document qu'il crée, où seule la structure spécifique du document et son contenu sont conservés. Ces fichiers contiennent la *représentation pivot* des documents, c'est à dire la représentation commune à toutes les applications qui doivent travailler sur les documents. C'est cette

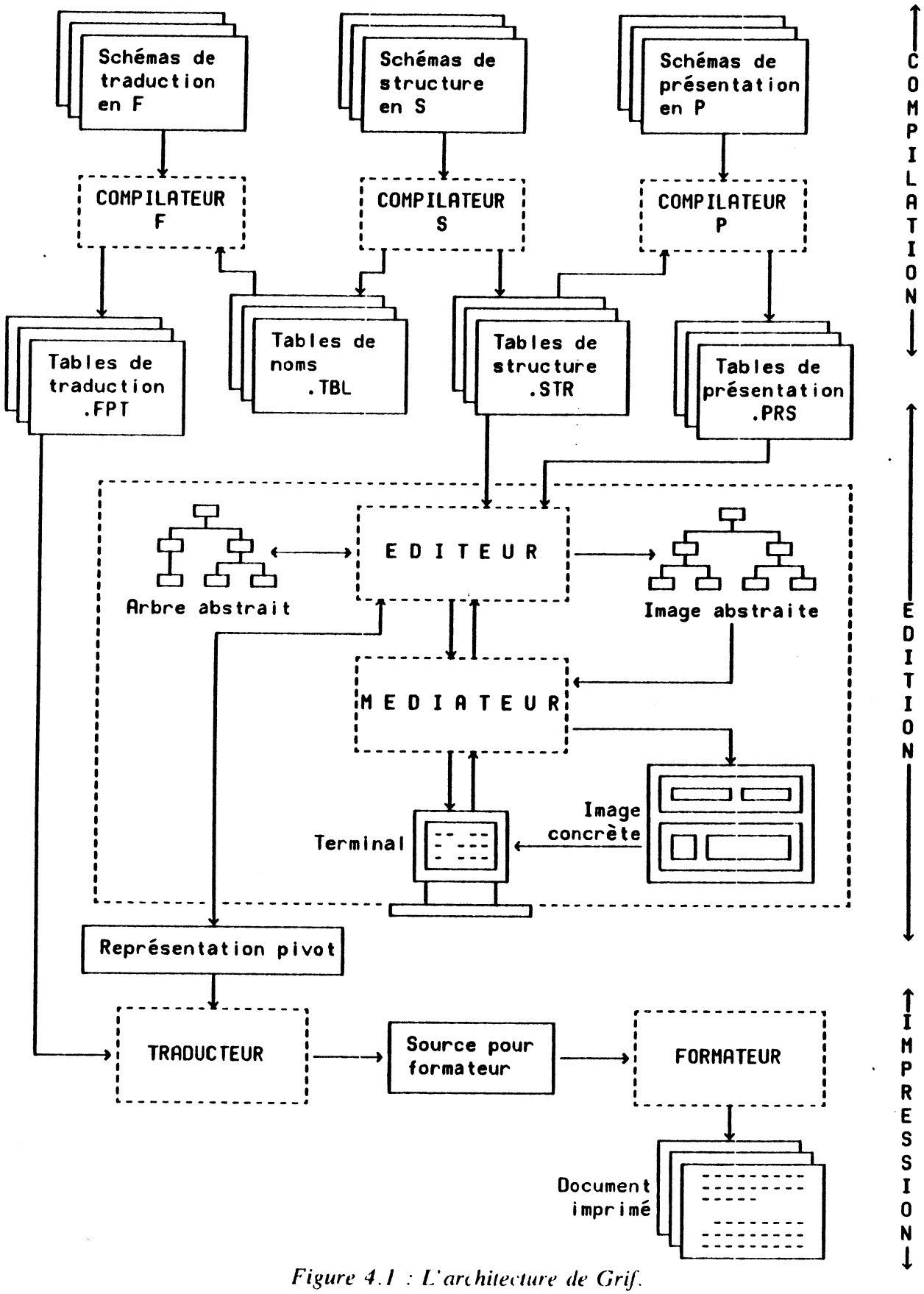


Figure 4.1 : L'architecture de Grif.

représentation qui est utilisée lorsqu'un document est chargé pour être édité par Grif ou lorsqu'il est sauvé après modification.

L'impression est réalisée indépendamment de l'édition des documents, par deux programmes. Le premier, le **traducteur**, transforme la représentation pivot d'un document en une description de ce même document dans le langage d'un formateur. Cette traduction est effectuée en suivant les *tables de traduction* spécifiques à la classe du document (et aux classes des objets qu'il contient) et au formateur qui doit être utilisé. Le fichier ainsi produit par le traducteur est ensuite soumis au **formateur** qui engendre le document imprimé.

Tous ces composants et les structures de données qu'ils manipulent sont présentés en détails dans les sections suivantes.

5. LES COMPILATEURS

Grif utilise trois langages et donc trois compilateurs. Pour éviter de développer pour chaque compilateur un analyseur lexical et un analyseur syntaxique spécifiques, ces outils sont partagés par les trois compilateurs. Un autre souci qui a guidé l'écriture de ces compilateurs est la souplesse d'évolution. Comme les concepts qui étaient à la base des langages n'avaient pas encore été éprouvés, les langages eux-mêmes n'étaient pas spécifiés d'une façon définitive au début de la réalisation ; ils ont d'ailleurs subi de nombreuses évolutions au fur et à mesure que l'expérience se développait. Il était donc important de pouvoir modifier les langages aisément et d'adapter de façon simple les compilateurs à chaque changement.

Il existe un certain nombre d'outils pour l'écriture des compilateurs, et notamment lex et yacc sous UNIX. Ces outils auraient certainement facilité la réalisation des compilateurs, mais ils n'étaient pas disponibles sur le PDP-11 qui a servi au développement initial. Cet inconvénient est maintenant devenu un avantage, puisque rien ne lie les compilateurs au système de développement, et qu'ils sont donc portables dans des environnements non-UNIX.

5.1. Les principes de construction des compilateurs

Chaque langage est décrit formellement à l'aide du formalisme qui a déjà été utilisé pour présenter les langages S et P. Ce formalisme est présenté dans l'annexe 1.

Avant de développer les trois compilateurs de Grif, nous avons développé un compilateur de ce méta-langage M qui, à partir de la description en M d'un langage, produit une table de règles syntaxiques. Ce sont ces règles qui sont ensuite utilisées par les compilateurs pour guider leur analyse syntaxique. Toutes ces tables étant construites de la même façon, le même programme est utilisé par tous les compilateurs (y compris le méta-compilateur) pour effectuer l'analyse syntaxique. Les compilateurs se distinguent les uns des autres uniquement par la partie de génération de code, qui est en fait une génération de tables. Ce principe de construction des compilateurs est résumé par la figure 4.2 qui présente le compilateur du langage S.

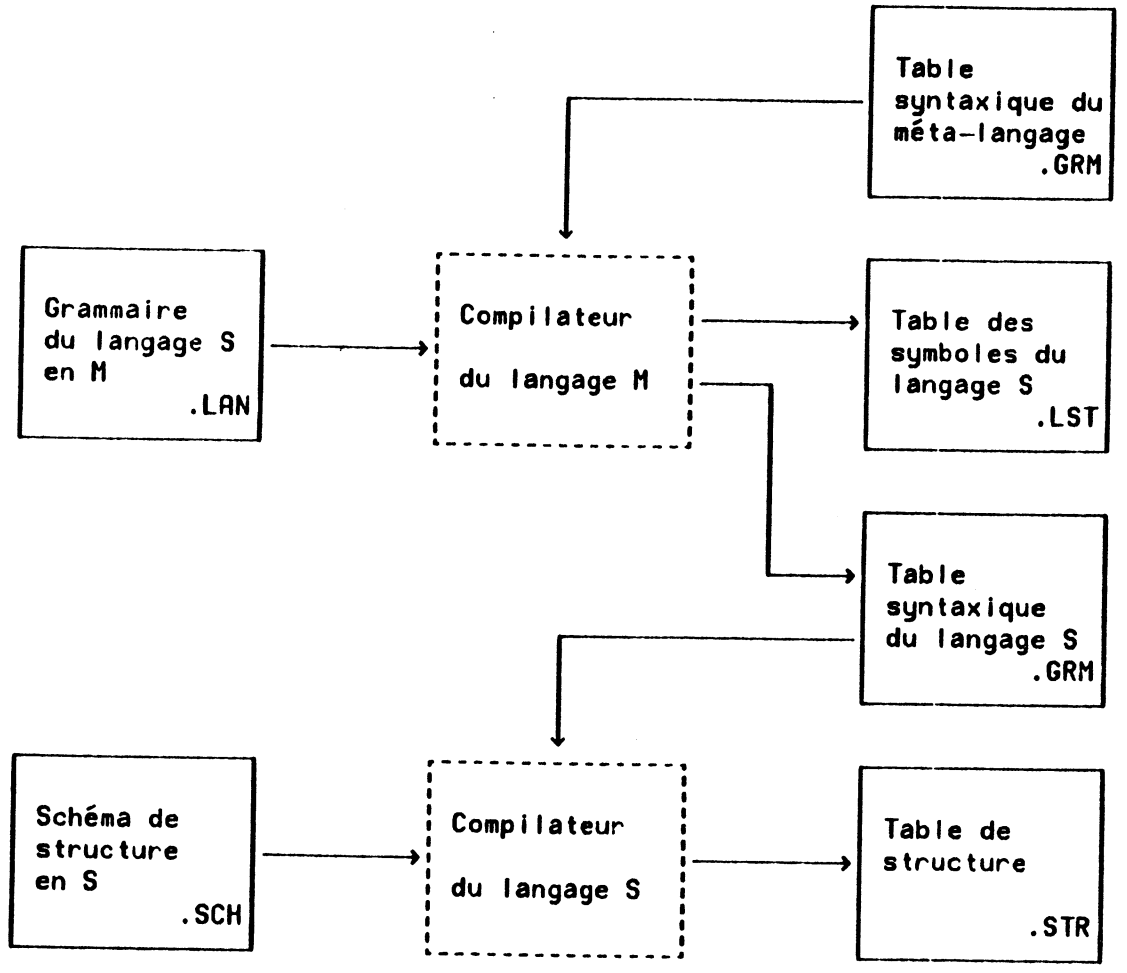


Figure 4.2 : Principe de fonctionnement des compilateurs.

Le compilateur du méta-langage M est lui-même construit de la même façon. Le mécanisme a été amorcé en écrivant "à la main" sa table syntaxique. Ce compilateur M lit le programme à compiler (la grammaire d'un langage, S pour la figure 4.2) qui se trouve dans un fichier de type .LAN. Il utilise sa propre grammaire, sous forme de table syntaxique. Il produit deux fichiers :

- Le fichier de type .LST est une liste imprimable de tous les symboles définis dans la grammaire et de tous les mots-clés du langage compilé, avec un numéro associé à chaque symbole et à chaque mot-clé.
- Le fichier de type .GRM est la table syntaxique de la grammaire du langage traité.

Le fichier .LST est utilisé ensuite pour écrire le générateur de code du compilateur. En effet, lorsqu'il lit un programme à compiler (par exemple un schéma de structure), l'analyseur syntaxique appelle le générateur de code pour chaque symbole rencontré, en lui passant le numéro du symbole (tel qu'il est défini dans la liste .LST) ainsi que le numéro de la règle de grammaire qui a été appliquée. Grâce à la liste des symboles et des règles (fichier .LST), le programmeur connaît les valeurs des paramètres que recevra le générateur de code qu'il écrit

5.2. Le compilateur de schémas de structure

Le compilateur des schémas de structure a été réalisé selon le principe que l'on vient de voir. A partir d'un fichier de type .SCH contenant la description en S d'une structure générique, et guidé par la table syntaxique du langage S, il produit, dans un fichier de type .STR, une table de structure directement utilisable par l'éditeur Grif.

La table de structure contient essentiellement les définitions des attributs et les règles de structuration des objets de la classe. Chaque définition d'attribut est constituée d'un nom (le nom de l'attribut) et de la liste de ses valeurs possibles (une liste de noms). Chaque règle de structuration contient :

- un nom, qui est le nom du type défini par la règle de structuration,
- le constructeur de ce type : identité, liste, agrégat, choix, référence ou structure externe,
- un ou plusieurs numéros d'entrées de la table, selon le constructeur, indiquant les types des éléments qui interviennent dans le constructeur,
- les attributs imposés au type, s'il y en a (voir l'instruction WITH du langage S).

En plus des contrôles syntaxiques assurés systématiquement par l'analyseur syntaxique, le compilateur de schémas de structures effectue certains contrôles d'ordre sémantique, de façon à préparer le travail de l'éditeur et à informer l'utilisateur de certaines erreurs. Ainsi, une fois la table de structure créée à partir du schéma de structure, celle-ci est analysée par le compilateur et complétée.

- Tous les types qui ne sont utilisés par aucune règles et qui ne sont ni des éléments associés, ni des unités exportées, sont signalés à l'utilisateur comme étant erronés. En revanche, on vérifie que les éléments déclarés comme associés ne figurent que dans des règles Référence. Si ce n'est pas le cas, une erreur est signalée. De plus, le compilateur ajoute une règle définissant un nouveau type d'élément qui est une liste d'éléments du type de ces éléments associés. On verra que cette règle définit pour l'Editeur la racine de l'arbre abstrait des éléments associés de ce type.

- Les types qui interviennent dans au moins une règle mais ne sont pas définis dans la table sont considérés comme des natures définies dans des schémas de structure externes qui seront importés lors de l'édition d'un document. Une entrée est ajoutée à la table pour chacun de ces types, avec le constructeur "structure externe".

- Les types récursifs, c'est à dire qui se construisent à partir d'eux-mêmes (exemples : une section contient des sections de niveau inférieur, une expression mathématique contient des expressions mathématiques...), sont systématiquement détectés et un indicateur "récursif" leur est ajouté dans la table. Cela évitera à l'éditeur de créer indéfiniment des sections ou des expressions mathématiques...

- Certains symboles ne sont introduits dans un schéma de structure que pour en alléger l'expression, notamment pour éviter de définir plusieurs fois une structure qui intervient en plusieurs endroits du même schéma. Ces symboles sont détectés et signalés à l'utilisateur, de façon qu'il sache qu'il est inutile de définir des règles de présentation ou de traduction pour ces types d'élément, qui n'apparaîtront jamais dans les documents. Le compilateur vérifie également qu'il n'y a pas de constructeur Référence qui pointe sur ces types d'éléments. A titre d'exemple, les deux extraits de schémas de structure suivants sont équivalents (l'éditeur construira les mêmes structures) :

(1)

```
Résumé =          LIST OF (Paragraphe);
Corps_Section =   LIST OF (Paragraphe);
```

(2)

```
Résumé =          Suite_Paragraphes;
Corps_Section =   Suite_Paragraphes;
Suite_Paragraphes = LIST OF (Paragraphe);
```

Dans le schéma (2), `Suite_Paragraphes` est un de ces symboles auxquels ne correspond aucun type d'élément dans le document.

Toutes ces déductions faites par le compilateur sont signalées à l'utilisateur de façon qu'il puisse vérifier qu'elles ne sont pas le résultat d'erreurs de sa part. En effet, une simple

faute de frappe peut conduire le compilateur à considérer qu'un type est une nature externe (le type erroné n'est pas défini dans le schéma).

En plus de la table de structure, le compilateur de schémas de structure engendre un fichier de type .TBL, qui est en fait une table simplifiée ne comportant que les noms de types et d'attributs, et qui permet au compilateur de règles de traduction de connaître le numéro (le rang dans la table) de chaque type ou attribut.

5.3. Le compilateur de schémas de présentation

Le compilateur de schémas de présentation est construit de la même façon que le compilateur de schémas de structure. Cependant, de façon à faire le lien entre la présentation et la structure, il utilise les tables de structure construites par le compilateur de schémas de structure. Il faut donc compiler le schéma de structure d'une classe avant de compiler un schéma de présentation de cette classe.

Le compilateur P produit une table des règles de présentation dans un fichier de type .PRS. Cette table est plus complexe que celle engendrée par le compilateur de schémas de structure, puisque les schémas de présentation contiennent des informations plus variées. Les règles de présentation sont regroupées en chaînes, avec une chaîne pour chaque ensemble de règles attaché à un type d'élément, un attribut ou une boîte de présentation. Sans entrer dans les détails, on peut indiquer les principaux éléments d'une table de présentation :

- La liste des noms des vues déclarées.
- La liste des compteurs de numérotation, avec pour chaque compteur les opérations de comptage qui doivent lui être appliquées et les types des boîtes de présentation dans lesquelles la valeur du compteur apparaît.
- La liste des constantes de présentation, avec pour chacune son type et sa valeur. Cette liste contient aussi bien les constantes définies dans la section CONST du schéma de présentation que les constantes définies dans les variables.
- La liste des variables, chaque variable étant formée d'une suite de fonctions. Cette liste contient aussi bien les variables définies dans la section VAR du schéma de présentation que les variables définies dans les règles Content.
- La liste des types de boîte de présentation, avec, pour chaque type de boîte, son nom, son contenu (une variable ou une constante appartenant à l'une des deux listes précédentes), et la chaîne des règles de présentation correspondant au type de boîte.

- La chaîne des règles de présentation par défaut.
- La liste des règles de présentation des attributs : pour chaque valeur de chaque attribut défini dans le schéma de structure, la chaîne des règles de présentation correspondantes.
- La liste des règles de présentation des éléments : pour chaque type d'élément défini dans le schéma de structure, une chaîne de règles de présentation.

Chaque chaîne de règles a une longueur variable. Seule la chaîne des règles de présentation par défaut contient nécessairement des règles définissant tous les paramètres de présentation. Les autres chaînes de règles ne contiennent que les différences par rapport aux règles par défaut. Elles sont vides lorsque toutes les règles par défaut doivent être appliquées.

Les correspondances entre les chaînes de règles d'une table de présentation et les types ou les attributs de la table de structure auxquels ces règles sont associées sont assurées par l'ordre des listes : la chaîne des règles de présentation associées à un type d'élément à le même rang dans la liste des règles de présentation des éléments que le type dans la liste des règles de structure.

De même que les schémas de structure, les schémas de présentation sont contrôlés à la compilation. Le compilateur vérifie notamment que le même paramètre de présentation n'est pas défini plusieurs fois dans la même chaîne, que tous les paramètres sont définis dans les règles par défaut, ou encore que les différentes règles d'une même chaîne ne sont pas incohérentes. C'est également lors de la compilation qu'est associée à chaque compteur la liste des types de boîtes de présentation qui utilisent le compteur. Cette liste facilitera le travail de l'Editeur lors des renumérotations après ajout ou suppression d'éléments numérotés.

6. L'INTERFACE EDITEUR-MEDIATEUR

Avant de détailler le fonctionnement de l'éditeur, guidé par les tables de structure et de présentation, nous présentons l'interface entre ses deux principaux constituants, l'Editeur et le Médiateur. C'est cette interface qui garantit une bonne séparation des deux niveaux de traitement : les manipulations de structure d'un côté (l'Editeur), les manipulations de contenu, l'affichage et le dialogue utilisateur de l'autre (le Médiateur). Cette séparation des taches présente de multiples avantages :

- Elle débarasse l'Editeur des fonctions interactives de plus bas niveau : saisie et édition du texte caractère par caractère, saisie des commandes et de leurs paramètres.

- Elle rend l'Editeur indépendant du matériel et lui offre le moyen d'exprimer les images à afficher à un niveau d'abstraction élevé, et non pas sous la forme de commandes destinées à un appareil, fût-il standard ou virtuel.
- Elle autorise l'utilisateur à adapter le comportement de l'interface à ses propres besoins, sans pour autant modifier l'application, essentiellement représentée par l'Editeur.

Le Médiateur offre à l'Editeur un ensemble de services concernant la gestion des fenêtres, l'affichage des messages, la mise à jour et l'affichage des images de documents, la sélection, la gestion des menus et commandes et la saisie du contenu.

L'ensemble Editeur-Médiateur ne constitue qu'un seul processus au sens du système sous lequel Grif s'exécute. Il ne nous a pas semblé nécessaire d'introduire d'asynchronisme entre les deux composants, puisque tous les traitements sont séquentiels et que les machines auxquelles Grif est destiné sont pour la plupart des mono-processeurs. Les communications entre l'Editeur et le Médiateur se font donc par appel de procédure, chaque composant appelant une procédure spécifique de l'autre chaque fois qu'il requiert ses services. Lors de l'appel d'une procédure, certains paramètres sont passés, mais les deux composants partagent de plus une même représentation des documents qu'ils manipulent, l'*image abstraite*. Pour garantir une certaine cohérence de cette structure de données partagée, seul l'Editeur a le droit de la modifier, le Médiateur ne pouvant que la lire.

6.1. Les images abstraites

Plusieurs images abstraites peuvent coexister. Il y en a une pour chaque vue qui a été créée par l'utilisateur et donc au moins une par document en cours de traitement.

Le Médiateur affiche chaque vue dans une fenêtre. La vue est un concept de l'Editeur. C'est lui qui connaît les relations entre les vues, qui sait que telle et telle vues sont liées parce qu'elles représentent le même document. Le Médiateur, lui, voit seulement un ensemble de fenêtres, avec une image abstraite à afficher dans chaque fenêtre. Il n'établit aucune relation entre les fenêtres qu'il gère ou entre les images abstraites qui leur sont associées. C'est à l'Editeur d'assurer la cohérence entre les différentes vues en mettant à jour les images abstraites et en les faisant réafficher par le Médiateur lorsque c'est nécessaire.

Chaque image abstraite décrit une image qui doit être affichée dans une fenêtre par le Médiateur. Que ce soit pour éditer un nouveau document ou pour visualiser une vue supplémentaire demandée par l'utilisateur pour un document en cours de traitement, chaque fois qu'il crée une nouvelle vue, l'Editeur demande au Médiateur de créer une

nouvelle fenêtre. Celui-ci demande à l'utilisateur de déterminer, avec la souris, la position et les dimensions de la fenêtre sur l'écran. Une fois la fenêtre créée, l'Editeur construit l'image abstraite de la partie de document qui doit être visualisée dans cette fenêtre, puis il demande au Médiateur d'afficher cette image abstraite. Ensuite, lorsqu'une modification sera apportée au document, provoquant un changement d'une image abstraite, c'est l'Editeur qui modifiera l'image abstraite et demandera au Médiateur de refléter cette modification sur l'écran.

Une image abstraite décrit une image indépendamment du support d'affichage. Ainsi, aucune position ou dimension absolue n'est indiquée, mais seulement des relations entres les positions et les dimensions des différents éléments de l'image (les boîtes que le Médiateur construira). De la même façon, les attributs typographiques (police, style, corps, graisse, etc...) sont donnés de façon relative, ce qui permet au Médiateur de les traduire de la façon la plus appropriée aux possibilités de l'appareil et aux choix de l'utilisateur. Pour la même raison, toutes les dimensions qui apparaissent dans une image abstraite sont données dans une unité de mesure abstraite, la même que dans les schémas de présentation : un dixième du corps des caractères.

Une image abstraite est un arbre constitué d'éléments appelés *pavés*. Chaque pavé est l'abstraction d'une *boîte* qui sera créée par le Médiateur. Seuls les pavés correspondant aux boîtes des éléments structurés et aux boîtes de présentation se retrouvent dans les images abstraites. Les boîtes de mise en lignes sont créées par le Médiateur et ne sont pas connues de l'Editeur, qui se contente d'indiquer au Médiateur les règles pour les construire. En particulier, si un paragraphe contient du texte, l'Editeur est incapable de décider du nombre de boîtes lignes que ce texte occupera puisque cela dépend notamment de la taille de chaque caractère et de la largeur de la fenêtre, toutes sortes de paramètres dont seul le Médiateur connaît la valeur. L'Editeur est donc incapable de donner une représentation, même abstraite, de ces boîtes lignes. En revanche, l'Editeur sait qu'il y aura une boîte paragraphe, et il sait comment elle doit être affichée, puisqu'il dispose des tables de présentation ; il est donc en mesure de créer un pavé pour le paragraphe, dans lequel il indique les paramètres de mise en lignes du contenu.

La structure d'arbre d'une image abstraite traduit les relations topologiques entre les boîtes correspondant aux pavés. Chaque pavé de l'arbre correspond à une boîte qui englobera toutes les boîtes correspondant à ses pavés fils : aucune des boîtes englobées ne dépassera de la boîte englobante. En revanche les positions relatives entre les boîtes englobées au premier niveau par une même boîte ne sont pas indiquées par la structure arborescente de l'image abstraite : chaque pavé porte lui-même des indications sur la position que doit occuper sa boîte par rapport à la boîte englobante ou aux boîtes de ses pavés frères.

Un pavé possède donc deux règles de positionnement, une pour le sens vertical et une autre pour le sens horizontal. Ces deux règles sont analogues à celles qu'on peut exprimer dans un schéma de présentation, à la différence que, dans l'image abstraite, les relations de position ont été évaluées et donc un pointeur sur un autre pavé indique précisément la boîte par rapport à laquelle on se positionne (la boîte de référence). Une règle de positionnement d'un pavé contient :

- le pointeur sur le pavé de la boîte de référence,
- le côté ou l'axe qui sert de repère dans la boîte de référence (RV ou RH),
- le côté ou l'axe qui détermine la position de la boîte positonnée (DV ou DH),
- la distance entre ces deux côtés ou axes (dv ou dh).

La figure 4.3 montre un exemple de positionnement de deux pavés frères. Dans le sens vertical, la boîte positionnée est la boîte B, qui se place de façon que son axe médian horizontal (DV) soit à la distance dv (positive, donc en-dessous) du bord supérieur (RV) de la boîte A (boîte de référence). Dans le sens horizontal, la boîte positionnée est la boîte A dont le côté droit (DH) est à une distance dh (négative, donc à gauche) du côté gauche (RH) de la boîte B (boîte de référence).

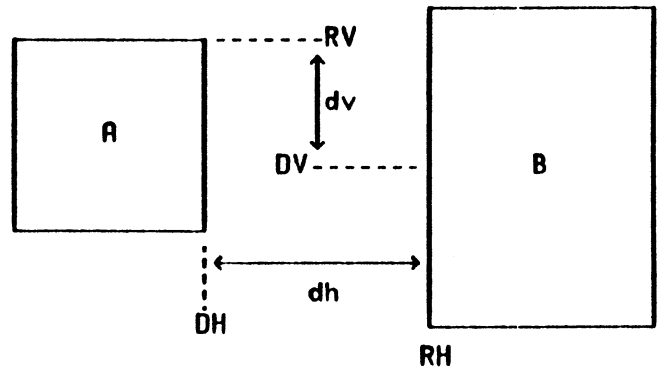


Figure 4.3 : Exemple de positionnement de deux pavés.

Comme pour la position, chaque pavé définit les deux dimensions, horizontale et verticale, de la boîte qu'il représente, chacune des dimensions étant indépendante de l'autre, mais décrite de la même façon : un pavé de référence (un pointeur sur le pavé), une valeur (un entier) et un indicateur de pourcentage (un booléen). Une dimension peut être exprimée de trois façons différentes :

- Absolue : la dimension a une valeur fixe imposée. Dans ce cas le pavé de référence et l'indicateur de pourcentage sont inutilisés et la valeur donne la dimension exprimée en unités de longueur.

- Relative : la dimension dépend de celle d'une autre boîte. La boîte dont elle dépend (dite boîte de référence) est celle qui correspond au pavé de référence. La relation peut être soit un pourcentage, soit une différence (c'est indiqué par l'indicateur de pourcentage). Dans le cas d'un pourcentage la valeur contient ce pourcentage p (qui peut être supérieur ou inférieur à 100) : la dimension définie est égale à p% de la même dimension de la boîte de référence. Dans le cas d'une différence, la valeur contient la différence d (qui peut être positive ou négative) : la dimension définie doit prendre la valeur de la même dimension de la boîte de référence, augmentée (ou diminuée, selon le signe) de d.

- Minimum : la dimension prend la valeur minimum pour que la boîte englobe juste l'ensemble des boîtes qu'elle contient, une fois ces boîtes dimensionnées et positionnées. Dans ce cas, le pavé de référence, l'indicateur de pourcentage et la valeur ne sont pas utilisés.

On voit que les dimensions et positions des pavés peuvent être calculées simplement, en appliquant les règles de présentation sur l'arbre abstrait décrivant l'organisation logique du document. L'essentiel de ce calcul se résume à la recherche du pavé de référence, à partir des indications fournies dans la règle de présentation.

Les positions et dimensions ne sont pas les seules informations portées par les pavés. Ceux-ci contiennent également des règles de positionnement des deux axes de référence (horizontal et vertical) de la boîte. Ces règles s'expriment de la même façon que les règles de positionnement, mais le pavé de référence est toujours un pavé fils ou le pavé lui-même et l'axe D est toujours l'axe de référence.

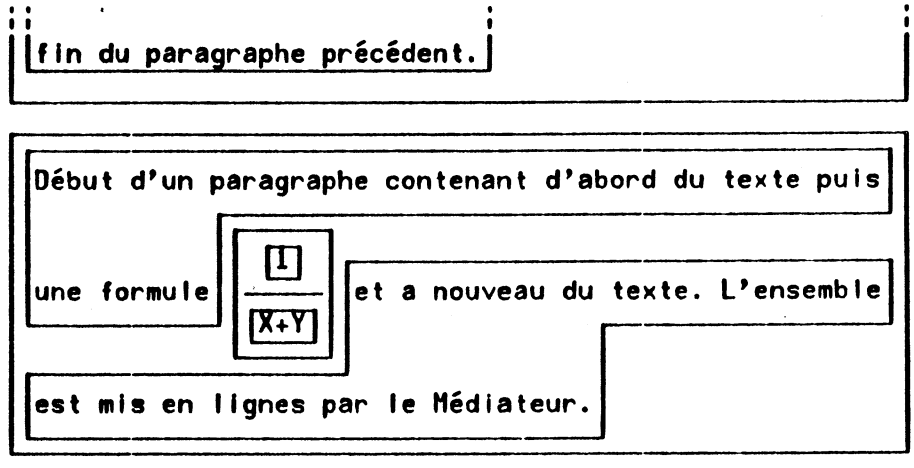
Un pavé contient également un pointeur sur l'élément de la structure logique dont il décrit l'image. Ce pointeur est notamment utilisé pour faire le lien entre une désignation sur l'écran (le Médiateur connaît le pavé correspondant à la boîte désignée) et un élément de la structure logique du document.

Les informations qui permettent au Médiateur de choisir un certain nombre d'attributs typographiques pour afficher le contenu de la boîte sont indiquées dans le pavé en terme relatifs, pour assurer l'indépendance de l'image abstraite vis à vis du matériel d'affichage. Les mêmes unités que dans les schémas de présentation sont utilisée pour exprimer le corps des caractères contenus dans la boîte, et leur mise en évidence. Tout comme les dimensions et positions, cela est calculé en appliquant les règles de présentation (souvent des règles d'héritage) sur l'arbre abstrait du document.

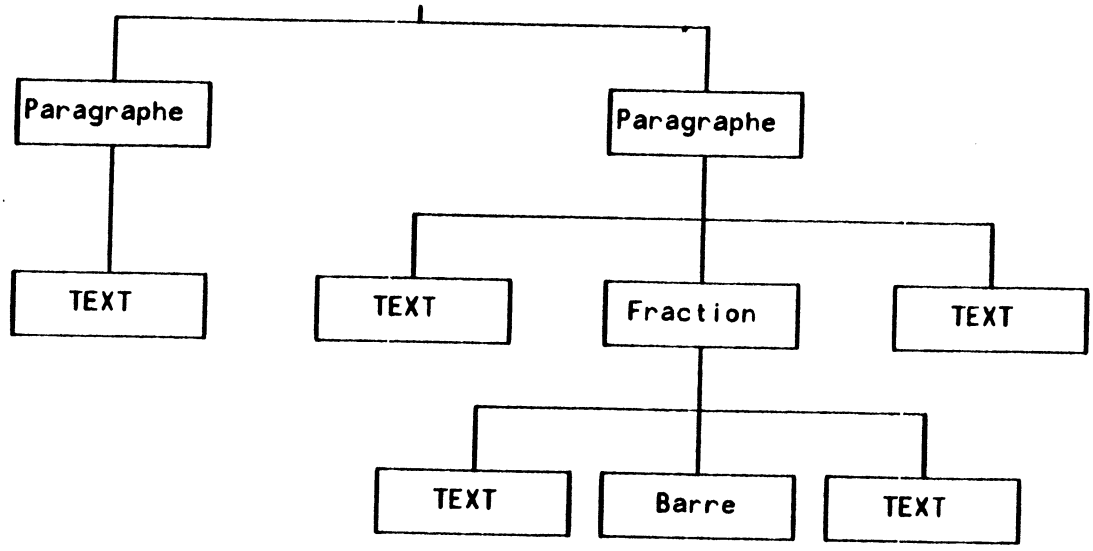
Trois autres informations indiquent au Médiateur comment il peut afficher ou modifier le contenu d'une boîte. La valeur du degré de visibilité est indiquée dans chaque pavé, ce qui permet au Médiateur d'afficher la vue avec le niveau de détail souhaité par l'utilisateur. Un pavé peut être marqué comme sélectionné, ce qui indique au Médiateur

qu'il doit l'afficher de telle façon que l'utilisateur voie la partie de l'image qui est sélectionnée. Enfin certains pavés peuvent être protégés en écriture, ce qui signifie que l'utilisateur ne pourra pas en modifier le contenu textuel. Cela est notamment utilisé pour les pavés des boîtes de présentation : pour le Médiateur, il n'y a pas de différence entre un pavé représentant une boîte de présentation et un pavé représentant un élément de la structure logique. Il y a seulement des pavés dont il peut modifier le contenu et d'autres dont le contenu ne peut pas être changé.

Le contenu d'un pavé peut être mis en lignes par le Médiateur. Dans ce cas les pavés de niveau inférieur ne sont pas positionnés explicitement les uns par rapport aux autres par des règles de positionnement, mais alignés implicitement les uns à la suite des autres sur leurs axes de référence horizontal, dans la limite de la largeur du pavé englobant. Lorsqu'une ligne ainsi constituée est pleine, une autre ligne est formée avec les pavés suivants, et ainsi de suite jusqu'à ce que tous les pavés fils du pavé mis en ligne soient traités. Au cours de ce processus de mise en lignes, les pavés fils du pavé mis en lignes, s'ils sont de type texte, peuvent être coupés et s'étendre sur plusieurs lignes. Les autres pavés fils (non terminaux ou terminaux non textuels) ne sont pas coupés. L'exemple ci-dessous le montre :



Ces boîtes sont construites par le Médiateur à partir de l'image abstraite suivante :



Dans cet exemple, les deux pavés Paragraphe sont mis en lignes. Il portent des règles de positionnement qui définissent leur place par rapport au paragraphe précédent et ils portent également l'indication "mise en lignes". Le deuxième paragraphe possède trois pavés fils, dont aucun ne porte de règle de positionnement, puisque leur position est déterminée par la mise en ligne. Le premier pavé fils est terminal de type texte et peut donc être coupé. Le deuxième n'est pas terminal ; il correspond à la fraction et contient deux pavés de texte et un de type graphique (la barre de fraction) qui sont positionnés explicitement les uns par rapport aux autres. Le troisième est un pavé terminal de type texte, qui peut être découpé en lignes comme le premier.

On voit que la mise en lignes peut produire des boîtes qui ne sont pas rectangulaires. C'est le cas en général des pavés de texte qui s'étendent sur plusieurs lignes, mais il n'y a pas d'autres boîtes non rectangulaires que les boîtes de texte mises en lignes.

Lorsqu'un pavé est mis en lignes, il porte, en plus de l'indication de mise en lignes elle-même, d'autres informations qui définissent plus précisément le mode de construction des lignes. L'Editeur déduit ces indications des règles de présentation de la boîte ligne (boîte de mise en page du schéma de présentation).

- L'*interligne* spécifie la distance minimum entre les axes de référence horizontaux (lignes de base dans le cas du texte) de deux lignes successives. Si une ligne comporte une boîte haute, comme la formule de l'exemple ci-dessus, l'interligne réel, calculé par le Médiateur, peut être supérieur à ce minimum, pour éviter les recouvrements.
- Le *cadrage* définit comment les boîtes lignes se placent horizontalement par rapport à la boîte englobante : sur le bord gauche (comme dans l'exemple ci-dessus), sur le

bord droit, centrées ou encore étirées jusqu'aux deux bords (sauf pour la dernière ligne qui n'est jamais étirée).

- Le *retrait* indique la distance entre le bord gauche de la première ligne et celui des lignes suivantes. Il peut être positif ou négatif, mettant la première ligne en retrait par rapport aux autres ou au contraire décalant à droite toutes les lignes par rapport à la première.

Toutes les informations que nous venons de décrire sont présentes dans tous les pavés. Les pavés constituant les feuilles de la structure arborescente contiennent une information complémentaire qui varie selon le type de feuille et indique son contenu. Les feuilles de type symbole ou graphique portent un seul caractère qui représente le symbole ou la figure élémentaire avec les mêmes conventions que dans le langage P. Les feuilles de type texte ou image pointent sur une chaîne de caractères qui est le texte lui-même ou le nom du fichier où se trouve l'image.

Pour résumer, chaque pavé d'une image abstraite contient donc :

- Quatre pointeurs vers d'autres pavés, décrivant ainsi la structure arborescente de l'image abstraite : pavé englobant, pavé précédent, pavé suivant, premier pavé englobé.
- Un pointeur vers l'élément correspondant de l'arbre abstrait.
- La hauteur et la largeur de la boîte à construire. Les deux dimensions sont spécifiées sous la même forme, mais elles sont indépendantes l'une de l'autre.
- La position horizontale et la position verticale de la boîte par rapport aux autres boîtes.
- Les positions des deux axes de référence par rapport à la boîte elle-même ou à une boîte englobée.
- Les attributs typographiques : mise en évidence, corps, sélection.
- Le niveau de visibilité.
- L'indication de mise en lignes et, si la mise en lignes est demandée :
  - l'interligne,
  - le cadrage horizontal des lignes,
  - le retrait de la première ligne.
- Pour les pavé terminaux, leur contenu.

On voit qu'un pavé comporte plusieurs pointeurs vers d'autres pavés, introduisant ainsi une structure complexe de graphe. Néanmoins, les quatre premiers pointeurs permettent de décrire la structure arborescente de l'image abstraite, et donc de traduire les relations d'englobement des boîtes. Les autres pointeurs introduisent des relations complémentaires à travers cet arbre. Ce sont les relations de position et de dimension. Ces relations sont toutefois limitées à des pavés voisins : les pavés frères, les pavés englobés ou le pavé englobant.

6.2. Les fenêtres

Le Médiateur peut créer et détruire des fenêtres sur demande de l'Editeur (fonctions CréationFenêtre et DestructionFenêtre). Lorsqu'une nouvelle vue doit être créée, pour afficher un nouveau document ou une vue supplémentaire d'un document déjà présent, c'est ce service que l'Editeur appelle, de même que lorsqu'il veut détruire une vue devenue inutile.

Mais les fenêtres peuvent également être détruites directement par l'utilisateur. Les gestionnaires de fenêtres offrent généralement cette possibilité. Dans ce cas, c'est le Médiateur qui est informé par le système de la destruction de la fenêtre, et c'est lui qui signale cet évènement à l'Editeur, en appelant la fonction DestructionVue de l'Editeur.

6.3. Les messages

Le Médiateur prend également en charge l'affichage de messages destinés à l'utilisateur. L'Editeur fournit le texte de ces messages et indique un niveau de sévérité lorsqu'il appelle la fonction Message du Médiateur. Le Médiateur affiche le texte dans la zone réservée à cet usage sur l'écran. Le niveau de sévérité lui indique s'il faut effectuer un simple affichage du message, ou s'il faut l'accompagner d'un signal sonore, ou encore demander que l'utilisateur confirme, par la frappe d'une touche, qu'il a bien pris connaissance du message. Ces effets sont ceux qui sont actuellement obtenus par défaut avec le Médiateur du Perq. Des effets différents peuvent traduire les niveaux de sévérité, par simple action de l'utilisateur sur le Médiateur et de façon transparente pour l'Editeur.

6.4. L'affichage et le réaffichage

L'affichage d'une nouvelle image et le réaffichage d'une image modifiée s'obtiennent de la même façon, en utilisant seulement deux fonctions basées sur les images abstraites : ModifierVue et AfficherVue. Affichage et réaffichage se font dans les fenêtres et le Médiateur ne connaît pas les relations entre les fenêtres. C'est l'Editeur qui demandera au Médiateur des réaffichages (indépendants pour le Médiateur) dans les différentes fenêtres associées aux vues d'un même document.

L'Editeur modifie (ou crée) l'arbre d'une image abstraite, et indique sur chacun des pavés les modifications effectuées : pavé créé, pavé mort, pavé dont les règles de présentation ou le contenu a changé. Les modifications peuvent porter sur plusieurs pavés et peuvent être de types différents (création, mort ou changement) : en effet la simple suppression d'un paragraphe, par exemple, provoque la mort du pavé paragraphe et de tous les pavés qu'il contient, mais provoque aussi des changements dans le pavé paragraphe suivant dont le positionnement doit se faire non plus par rapport au paragraphe supprimé, mais par rapport à celui qui le précède. Notons ici que ce sont les changements effectués sur les pavés qui sont signalés, même si, sur l'image finale, d'autres changements sont visibles. Pour reprendre l'exemple, seul le pavé du paragraphe suivant le paragraphe détruit est marqué comme changé, et les pavés des autres paragraphes suivants ne sont pas modifiés, bien que leur boîte sur l'écran soit déplacée vers le haut. Ce déplacement des autres parageraphes est évalué et effectué par le Médiateur qui utilise les paramètres de positionnement que ces pavés avaient déjà avant la suppression ; ces pavés ne changent donc pas.

Une fois l'image abstraite créée ou mise à jour, l'Editeur appelle le Médiateur (fonction ModifierVue) en lui signalant le pavé dont le sous-arbre contient toutes les modifications effectuées depuis la dernière mise à jour signalée au Médiateur (par ModifierVue) de la même image abstraite. Le Médiateur répercute alors ces modifications sur l'image concrète, sans toutefois l'afficher immédiatement ; ainsi plusieurs ensembles de modifications peuvent être effectués successivement par l'Editeur sans que l'image change sur l'écran. C'est l'Editeur qui décide du moment où cette image doit être réaffichée, lorsqu'il considère la série de modifications de l'image comme complète. Il demande alors au Médiateur (fonction AfficherVue) de visualiser l'image qui a été mise à jour. Notons que, lorsqu'un pavé doit être détruit, il est d'abord marqué "mort" par l'Editeur, cette modification est notifiée au Médiateur, puis, une fois qu'elle a été prise en compte par le Médiateur, le pavé est retiré de l'arbre par l'Editeur.

L'Editeur ne construit pas les images abstraites des documents dans leur totalité : elles seraient beaucoup trop volumineuses dès que les documents atteignent un volume de quelques pages. Il ne prépare que la partie (de fait un peu plus) qui sera effectivement

visible dans la fenêtre correspondante. Pour cela le Médiateur indique, lors de la création d'une fenêtre, le volume approximatif de l'image qu'il peut y afficher. Ce volume est exprimé en nombre de caractères moyens. Pour chaque image abstraite, l'Editeur engendre donc des pavés jusqu'à concurence de ce volume et il fait en sorte que le volume de l'image reste à peu près constant lors des modifications qu'il y apporte, en créant par exemple de nouveaux pavés lors de la destruction d'une partie du document.

L'Editeur n'ayant pas connaissance de la surface occupée sur l'écran par les pavés qu'il construit, il peut arriver que l'image abstraite corresponde à une image concrète beaucoup plus grande que la fenêtre (si le texte de cette image est très aéré ou si on utilise de gros caractères) ou au contraire trop petite (texte dense avec de petits caractères). Dans ce cas le Médiateur peut demander un ajustement du volume de l'image abstraite grâce aux deux fonctions AugmenterVolume et RéduireVolume. L'Editeur ajoute (ou retire) alors des pavés dans l'image abstraite, pour une variation de volume choisie par le Médiateur ; et cet ajout (ou retrait) peut se faire, au choix du Médiateur, au début ou à la fin de l'image abstraite. Ce sont également ces deux fonctions qui sont appelées par le Médiateur lorsque l'utilisateur fait défiler une image dans une fenêtre.

6.5. La sélection

L'Editeur et le Médiateur utilisent le principe de sélection pour définir de nombreux paramètres des commandes qu'ils traitent. L'utilisateur détermine d'abord la partie du document sur laquelle il veut agir (c'est la sélection), puis il déclenche une commande qui s'appliquera à la partie sélectionnée. La sélection est contrôlée alternativement par l'Editeur et par le Médiateur, mais jamais par les deux à la fois. Le Médiateur accède à l'intérieur des feuilles de la structure et c'est lui qui gère la sélection des caractères, des éléments graphiques, des images ou des symboles. L'Editeur, lui, traite la sélection des éléments structurés.

Le Médiateur peut visualiser la sélection courante de différentes façons, de manière transparente pour l'Editeur. Il peut afficher la partie sélectionnée en inversion video, ou sur un fond de couleur, ou en clignotant ou encore en traçant le contour de cette partie. Mais que la sélection soit contrôlée par l'Editeur ou par le Médiateur, elle est visualisée par le même moyen, et cette visualisation est assurée par le Médiateur.

Le Médiateur permet à l'utilisateur de sélectionner en pointant directement sur l'écran le début et la fin de la partie à sélectionner. Il l'autorise ensuite à modifier cette sélection en l'élargissant, en la réduisant ou encore en la changeant complètement. Ce type de sélection est géré par le Médiateur qui visualise lui-même la partie sélectionnée. Il se contente de signaler chaque changement de la sélection à l'Editeur (par appel de la fonction

SélectionCourante). Le Médiateur peut sélectionner de façon fine à l'intérieur des pavés. Il peut sélectionner un unique caractère ou une sous-chaîne de caractères dans un pavé, aussi bien qu'une suite de pavés complétée par une sous-chaîne du pavé de texte précédent et une sous-chaîne du pavé de texte suivant. L'utilisateur est ainsi libre de sélectionner comme il le ferait avec un système WYSIWYG.

S'il veut tirer parti de la structure du document, l'utilisateur doit sélectionner sous le contrôle de l'Editeur, puisque seul l'Editeur a la connaissance de la structure logique des documents. Dans ce cas, l'Editeur doit pouvoir demander au Médiateur de visualiser la sélection courante. Cette modification de l'image affichée est très différente de celles présentées dans la section précédente, puisqu'elle n'affecte pas les dimensions ni les positions des différents éléments d'image, mais seulement leur aspect visuel. Avec la fonction ChangeSélection l'Editeur peut sélectionner un pavé qui ne l'était pas, ou désélectionner un pavé qui était sélectionné. Les pavés concernés doivent évidemment faire partie d'une image abstraite et doivent déjà avoir été affichés par le Médiateur avant d'être sélectionnés. Si plusieurs pavés doivent être sélectionnés, l'Editeur appelle la fonction ChangeSélection pour chacun des pavés. Si des pavés de texte doivent être sélectionnés partiellement, la fonction PoseSélection permet à l'Editeur de ne sélectionner qu'une sous-chaîne d'un élément de texte. Cela est particulièrement utile pour les commandes de recherche de chaînes de caractères qui sont traitées par l'Editeur et qui aboutissent souvent à sélectionner un mot au milieu du texte d'un paragraphe.

Lorsque c'est l'Editeur qui contrôle la sélection, le Médiateur peut avoir besoin de savoir quel est le premier pavé de la sélection courante. Il dispose pour cela de la fonction MarqueInsertion. Cela lui permet notamment de savoir où doivent aller les caractères frappés au clavier (voir plus loin la saisie du contenu).

Pour autoriser l'utilisateur à passer d'un mode de sélection à l'autre très librement, le contrôle de la sélection doit être partagé entre l'Editeur et le Médiateur. La fonction RazSelection permet à l'Editeur de reprendre le contrôle de la sélection. Le Médiateur sait alors qu'il perd le contrôle de la sélection et que les commandes que l'utilisateur pourra appeler et qui portent sur la sélection devront être transmises à l'Editeur. Inversement, lorsque l'utilisateur souhaite sélectionner au niveau du Médiateur, celui-ci appelle la fonction AnnuleSélection pour indiquer à l'Editeur qu'il perd le contrôle de la sélection et qu'il n'y a plus de sélection courante pour lui. Les commandes portant sur la sélection seront à nouveau traitées par le Médiateur.

Pour l'utilisateur le basculement d'un mode de sélection à l'autre se fait très simplement. Lorsque la sélection est au niveau de l'Editeur, c'est à dire qu'un ou plusieurs éléments de structure sont sélectionnés, et que l'utilisateur souhaite sélectionner à l'intérieur d'une feuille, par exemple un simple mot, il lui suffit de pointer avec la souris sur le début de ce mot. Le Médiateur reprend alors le contrôle de la sélection et sélectionne le

caractère pointé. Pour basculer en sens inverse, l'utilisateur dispose de deux moyens complémentaires : il peut appeler le menu de sélection de l'Editeur, en pressant sur le bouton "Menu", ou simplement désigner sur l'écran, à l'aide de la souris, un point qui n'est pas dans un des pavés feuilles de l'image abstraite. Par exemple, pour sélectionner tout un paragraphe, il suffit de désigner, sur la dernière ligne, un point qui est au-delà du dernier caractère. Dans ce cas, le Médiateur appelle l'Editeur (fonction SélectionPavé ou ExtensionSélection, selon le bouton pressé) pour lui indiquer le pavé désigné par l'utilisateur, et c'est l'Editeur qui sélectionne l'élément de structure correspondant à ce pavé.

6.6. Les commandes

Le Médiateur prend en charge la saisie des commandes de l'utilisateur. Il propose pour cela deux moyens différents : les touches de fonctions (ou menus permanents) et les catalogues, et il laisse à l'Editeur le soin de choisir le moyen adéquat, selon la commande.

6.6.1. Les touches de fonction

Les touches de fonction déclenchent deux catégories de commandes : les commandes prédéfinies du Médiateur et les commandes définissables par l'Editeur.

Les **commandes prédéfinies** sont les quatre commandes génériques Insérer, Couper, Coller et Copier. Selon que la sélection est contrôlée par le Médiateur ou l'Editeur, elles sont traitées différemment. Lorsque la sélection est sous le contrôle du Médiateur ces commandes sont traitées par le Médiateur de la façon suivante :

- Insérer est sans action si la sélection porte sur du texte (pour insérer du texte, il suffit de le taper au clavier). Si c'est une image qui est sélectionnée, la commande Insérer permet de saisir le nom d'un fichier contenant une image, l'image venant s'afficher dans l'élément sélectionné. Si la sélection porte sur un élément graphique ou un symbole, Insérer demande la frappe d'un caractère identifiant l'élément graphique ou le symbole à mettre à la place de la sélection (l'image d'un clavier avec tous les symboles ou éléments graphiques est affichée pour faciliter la saisie).

- Couper sauve dans une mémoire tampon toute la partie sélectionnée (le contenu des feuilles uniquement) et la supprime de l'écran en réaffichant des pavés vides ou réduits.

- Coller insère devant la sélection courante le contenu de la mémoire tampon. Il s'agit encore d'une insertion à l'intérieur d'un pavé feuille.

- Copier sauve la partie sélectionnée, comme Couper, mais sans la supprimer du document.

Les modifications de contenu effectuées par ces commandes du Médiateur sont signalées à l'Editeur par la fonction NouveauContenu. Le Médiateur connaît notamment les attributs typographiques et les règles de mise en lignes des pavés et les applique au contenu ajouté par les commandes Insérer et Coller. Ainsi un texte, coupé dans un titre centré et affiché en grands caractères, est justifié et affiché en petits caractères lorsqu'il est collé dans un paragraphe, et cela sans que l'utilisateur ni l'Editeur aient à s'en soucier.

Si la sélection est sous le contrôle de l'Editeur, le Médiateur fait traiter la commande par l'Editeur en appelant la fonction correspondante : CommandeInsérer, CommandeCouper, CommandeColler, CommandeCopier. L'Editeur applique alors la commande à la sélection courante en faisant un traitement analogue à celui du Médiateur, mais sur la structure du document et non pas sur son contenu seulement. C'est notamment lors du traitement de ces commandes que l'Editeur demande au Médiateur des réaffichages.

Les **commandes définissables** par l'Editeur ne sont jamais traitées par le Médiateur qui se contente de les saisir et d'appeler l'Editeur pour le traitement, de la même façon que pour les commandes prédéfinies lorsque la sélection est contrôlée par l'Editeur. Pour cela le Médiateur appelle une fonction de l'Editeur, Commande, en lui donnant le numéro de la commande à traiter.

Toutes les commandes, prédéfinies ou définissables, sont vues et activées par l'utilisateur de la même façon. Elles sont affichées sur l'écran par le Médiateur sous la forme d'une suite de rectangles contenant un libellé qui indique l'action de la commande. Les libellés des quatre commandes prédéfinies sont "Insérer", "Couper", "Coller" et "Copier". Ceux des autres commandes sont définis par l'Editeur, grâce à la fonction LibelléCommande, qui associe un texte à un numéro de commande. Cette fonction peut être appelée plusieurs fois pour la même commande, de façon à en changer le libellé dynamiquement. Pour déclencher une commande l'utilisateur peut soit frapper une des touches de fonction du clavier, soit "cliquer" le rectangle correspondant sur l'écran. On peut donc considérer l'ensemble de ces commandes comme un menu affiché en permanence sur l'écran, et dont le contenu peut varier dans le temps.

6.6.2. Les catalogues

D'autres menus sont proposés, qui sont du type fugitif ("pop-up"). Ils doivent être demandés explicitement soit par l'utilisateur, à l'aide d'un bouton réservé à cet usage, soit par l'Editeur. Dès qu'il est demandé par l'utilisateur ou l'Editeur, le menu courant s'affiche à la position de la souris. C'est l'Editeur qui définit le menu courant, donnant

ainsi la liste des commandes valides à l'instant considéré. L'utilisateur peut choisir une entrée de ce menu soit en la "cliquant" avec la souris, soit en frappant une touche du clavier, associée à l'entrée choisie. Le menu disparaît alors de l'écran et le Médiateur transmet à l'Editeur le choix de l'utilisateur.

La notion de menu du Médiateur a été élargie pour permettre non seulement le choix parmi un ensemble limité de propositions, mais aussi la saisie de paramètres plus libres et de différents types. Cette notion de menu élargie est appelée *catalogue*. Grâce aux catalogues, l'Editeur peut demander à l'utilisateur, par l'intermédiaire du Médiateur :

- de sélectionner une entrée dans un menu,
- d'entrer une chaîne de caractères, avec une longueur maximum,
- d'entrer un entier, avec une valeur maximum,
- de désigner sur l'écran une fenêtre, une boîte (donc un pavé), un caractère ou encore un élément graphique.

Tous ces types de saisie se font en deux temps. L'Editeur définit d'abord les catalogues qu'il utilisera, puis il déclenche la saisie au moment où il a besoin d'un paramètre. La définition se fait à l'initialisation de l'Editeur, pour un certain nombre de catalogues qui ne changent pas et qui sont utilisés plusieurs fois. Pour d'autres catalogues, plus dynamiques, la définition a lieu juste avant l'activation, et, une fois la saisie effectuée, le catalogue est soit détruit soit conservé pour être modifié lors de l'appel suivant, s'il peut être réutilisé. Le Médiateur autorise en effet à tout instant la création de nouveaux catalogues, la modification des catalogues existants ou la suppression des catalogues devenus inutiles. Pour cela il fournit quatre fonctions :

- CatalogueElémentaire définit (ou redéfinit) un catalogue élémentaire qui permettra, selon son type, indiqué à l'appel de la fonction, de saisir une chaîne de caractères ou un entier, ou de désigner une fenêtre, un pavé, un caractère ou un élément graphique.
- CatalogueListe définit un catalogue liste (ce qui correspond à la notion usuelle de menu) en donnant le texte de chacune des entrées.
- ActiveCatalogue demande au Médiateur d'activer un catalogue (élémentaire ou liste) spécifié et de retourner le choix de l'utilisateur ou ce qu'il a entré, selon le type du catalogue. Ainsi le Médiateur est chargé de solliciter l'utilisateur, de saisir et éventuellement d'éditer ce qui est saisi, et de contrôler la validité de la saisie. Il ne retourne à l'Editeur qu'une valeur considérée comme correcte. L'activation d'un catalogue peut être immédiate ou non. Une activation immédiate affiche le catalogue sur l'écran, demande à l'utilisateur une saisie immédiate pour ce catalogue, et

retourne à l'Editeur la valeur saisie. Une activation différée laisse l'utilisateur décider du moment où il souhaite utiliser le catalogue. Il peut ainsi éditer le contenu de son document ou appeler des commandes par les touches de fonction avant d'activer le catalogue courant.

- DétruitCatalogue offre à l'Editeur la possibilité de supprimer un catalogue devenu inutile.

Qu'il s'agisse du menu permanent ou des catalogues, toutes les commandes peuvent être entrées indifféremment au clavier ou avec la souris, selon le choix de l'utilisateur.

6.7. La saisie du contenu

Le Médiateur ne saisit pas seulement les commandes et leurs paramètres, à l'aide des touches de fonction ou des catalogues, il saisit aussi le contenu des documents en cours d'édition. Cette saisie s'apparente à celle qui est faite par les systèmes WYSIWYG : le texte est entré directement à sa place dans l'image du document. Chaque caractère frappé s'insère devant la sélection courante et est pris en compte pour la mise en page de l'ensemble de la fenêtre où il s'insère. Ainsi, un caractère supplémentaire peut faire déborder une ligne. Le bout de la ligne est alors reporté au début de la ligne suivante, qui peut à son tour déborder, et ainsi de suite jusqu'à provoquer l'extension verticale du paragraphe, ce qui repousse le paragraphe suivant. Dans une formule mathématique les conséquences peuvent être encore plus importantes, comme on l'a vu avec Edimath.

Toutes ces modifications de l'image du document sont effectuées sans intervention de l'Editeur, par le Médiateur seul, qui est guidé par les relations de dimension et de position exprimées sur les pavés : le Médiateur fait en sorte que tous les paramètres de présentation attachés aux pavés par l'Editeur soient toujours respectés. C'est encore lui qui affiche les caractères frappés au clavier en utilisant la police voulue, qui lui est indiquée par les paramètres taille et mise en évidence du pavé de texte.

La saisie du contenu ne se limite pas à la saisie de texte. Le Médiateur saisit tous les contenus : symboles, éléments graphiques, image. On a déjà vu, avec la commande générique Insérer, comment la saisie des éléments non-textuels pouvait être faite. La commande Insérer est surtout utile pour les utilisateurs peu habitués, puisqu'elle guide la saisie des symboles et des éléments graphiques avec une image de clavier ou qu'elle demande explicitement un nom de fichier, dans le cas d'un élément image. Si on connaît les conventions de frappe du système, il suffit de sélectionner le pavé dont on veut entrer le contenu et de frapper la touche du clavier correspondant au contenu souhaité.

Les modifications de contenu, bien que prises en charge par le Médiateur, intéressent aussi l'Editeur puisqu'elles affectent le contenu des feuilles de l'arbre abstrait. Ces modifications sont donc transmises à l'Editeur par l'appel de la fonction NouveauContenu. Cette fonction n'est pas appelée à chaque caractère frappé par l'utilisateur, mais seulement lorsqu'une chaîne de caractères semble terminée, c'est à dire lorsqu'une touche de fonction ou un bouton de la souris est pressé. Pour éviter de dupliquer le contenu du document, les tampons de texte sont partagés par l'arbre abstrait et l'image abstraite. Le Médiateur peut donc ajouter des tampons de texte dans l'arbre abstrait (ou en retirer sur la commande Couper). Pour cela, il utilise les services de l'Editeur, qui alloue et libère ces tampons (fonctions AllouеBufferTexte et LibèreBufferTexte).

De façon à simplifier l'interface utilisateur, il est admis que des caractères soient frappés alors qu'un pavé qui ne correspond pas à un élément de base est sélectionné. Cela est interprété par le Médiateur comme une volonté de l'utilisateur de créer un pavé de texte à l'intérieur du pavé sélectionné. Toutefois le Médiateur ne peut pas prendre l'initiative de créer des pavés. Il demande donc à l'Editeur (fonction CréeFeuille) de créer un élément de texte comme descendant de l'élément correspondant au pavé sélectionné. Si le schéma de structure l'autorise, l'Editeur crée la descendance voulue et rend au Médiateur le pavé de texte correspondant. La saisie peut alors avoir lieu normalement, sous le seul contrôle du Médiateur.

La fonction CréeFeuille est également appelée par le Médiateur lorsque l'utilisateur cherche à coller un élément de base (texte, symbole, image, élément graphique) dans un pavé correspondant à un élément structuré. Dans ce cas le Médiateur demande à l'Editeur de créer pour cet élément la descendance qui contient une feuille du type voulu.

6.8. Les autres appels du Médiateur

Deux fonctions supplémentaires sont offertes par l'Editeur au Médiateur. La première, InitEditeur, initialise l'Editeur. Elle est appelée par le Médiateur dès qu'il a lui-même terminé son intialisation. Outre ses initialisations propres, l'Editeur effectue dans cette fonction la création de ses catalogues permanents, ce qui permettra ensuite au Médiateur, et donc à l'utilisateur, d'appeler les principales commandes de l'Editeur, comme la création d'un nouveau document ou la lecture d'un document existant.

La deuxième fonction indique au Médiateur le type de l'élément auquel correspond un pavé donné. Le Médiateur en effet n'a pas directement accès à l'arbre abstrait et ne peut donc connaître ce type directement. En revanche, il en a besoin pour indiquer à l'utilisateur le type correspondant au pavé où se trouve la sélection courante, ce qui permet

notamment d'identifier certains pavés lorsqu'ils sont vides et donc peu reconnaissables. Le Médiateur affiche toujours le type des pavés qu'il sélectionne.

6.9. Quelques remarques sur l'interface Médiateur-Editeur

Bien que le Médiateur n'ait été utilisé jusqu'à présent qu'avec l'Editeur, il a été conçu avec un objectif plus large. Il s'agissait de définir et d'expérimenter un outil d'interface homme-machine, utilisable par plusieurs applications. Dans ce contexte, un tel outil présente les mêmes avantages qu'avec l'Editeur de Grif :

- Il débarasse les applications de la tâche de saisie du texte ainsi que de la saisie des commandes et de leurs paramètres.
- Il rend les applications indépendantes du matériel, et leur offre le moyen d'exprimer les images à afficher à un niveau assez élevé.
- Il autorise l'utilisateur à adapter le comportement de l'interface à ses propres besoins, sans pour autant modifier l'application.
- Il fournit enfin des interfaces uniformes pour plusieurs applications, en utilisant les mêmes moyens de communication avec l'utilisateur.

La partie la plus délicate dans la conception du Médiateur a sans doute été la définition du niveau correct de représentation des images abstraites. L'approche appareil virtuel est souvent utilisée pour rendre les applications indépendantes des terminaux. Elle nous a semblé d'un niveau trop bas puisque nous voulions que l'application (l'Editeur dans le cas de Grif) effectue les principaux choix de présentation, mais pas la mise en œuvre de ces choix. Ainsi, des fonctions comme la coupure du texte en lignes, la justification ou le centrage des lignes, et l'exécution dynamique de ces fonctions selon les modifications de contenu apportées par l'utilisateur, nous ont semblées d'un intérêt suffisamment général pour être réalisées au niveau des outils d'interface. De cette façon, toutes les applications qui affichent du texte modifiable par l'utilisateur, et elles sont nombreuses, peuvent être débarassées de ces fonctions.

Plusieurs propositions ont déjà été faites dans ce sens, et des normes émergent dans ce domaine. Les applications graphiques ont déjà la norme GKS [GKS] qui prend bien en compte l'aspect dynamique des images, mais qui n'offre pas, dans le domaine textuel, les services voulus. D'autres propositions sont plus orientées vers le texte, comme notamment les langages de description de page [Reid 86] PostScript [Adobe] ou Interpress [Sproull] [Xerox] qui offrent bien toutes les possibilités nécessaires pour l'affichage de document mais qui ne semblent pas bien adaptés à l'aspect dynamique des applications interactives. Il

faut toutefois noter un récent développement de PostScript vers des utilisations interactives [Sun].

Une autre proposition est d'un grand intérêt pour la description de documents dynamiques, c'est Interscript [Ayers]. Bien qu'il ne semble pas qu'Interscript devienne un standard, il comporte de nombreux aspects intéressants. Notamment le modèle selon lequel le contenu d'un document est découpé en lignes ou en colonnes, en "versant" le texte comme un liquide dans un récipient au contour déterminé, est assez proche de celui utilisé dans le Médiateur pour mettre le contenu en lignes dans des boîtes devant respecter certaines contraintes. De même l'idée de définir la mise en page par des contraintes et des relations entre les différents éléments se retrouve dans Interscript.

Un autre bénéfice du Médiateur est le réaffichage incrémental. Lorsqu'une image doit être modifiée, l'Editeur n'a qu'à mettre à jour l'image abstraite, ce qui est pour lui une tâche simple. C'est le Médiateur qui évalue toutes les répercussions des modifications sur l'ensemble de l'image et qui réaffiche dans l'image réelle la plus petite partie possible. Ainsi, l'application est déchargée de cette évaluation, et toutes les applications qui affichent des images modifiées dynamiquement peuvent partager ces fonctionnalités.

7. L'EDITEUR

L'Editeur se présente comme une application contrôlée par le Médiateur, et à travers lui, par l'utilisateur. Il est initialisé par le Médiateur et appelé par lui chaque fois que l'utilisateur entre une commande explicite (choix dans un menu, touche de fonction) ou implicite (modification de contenu d'un élément). Dans le traitement de ces commandes, l'Editeur peut solliciter les services du Médiateur, par exemple en créant une fenêtre ou en modifiant une image abstraite et en la faisant réafficher.

Avant de décrire de façon plus précise le comportement de l'Editeur, nous présentons les principales structures de données sur lesquelles il travaille et qui déterminent très directement les traitements qu'il peut effectuer.

7.1. Les structures de données

Les principales structures de données manipulées par l'Editeur sont (voir figure 4.4) :

- les tables de structure qui décrivent les structures génériques d'après lesquelles sont construits les documents en cours de traitement,

- les tables de présentation qui définissent les règles de présentation attachées aux types d'éléments et aux attributs définis dans les tables de structure,
- les arbres abstraits qui représentent la structure spécifique des documents en cours de traitement,
- les images abstraites qui spécifient les images de ces documents que le Médiateur doit afficher.

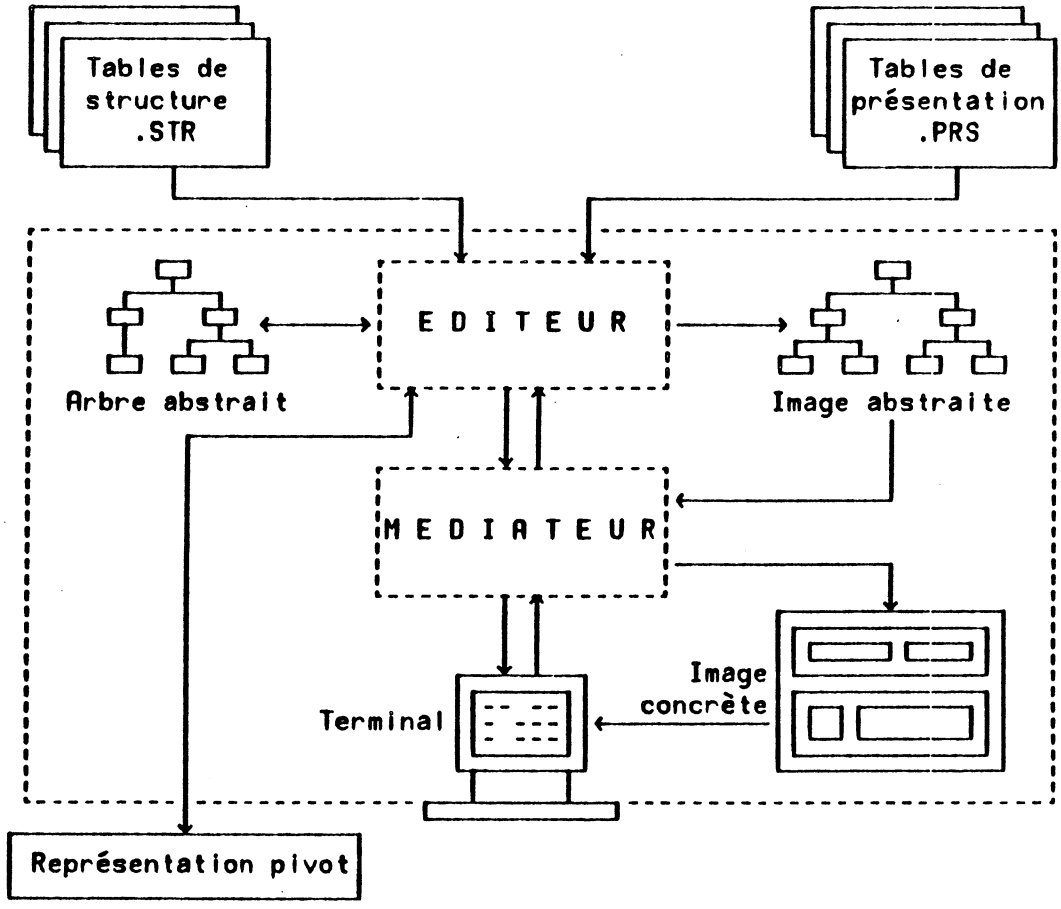


Figure 4.4 : Les structures de données de l'éditeur Grif.

Le contenu des tables de structure et de présentation ainsi que celui des images abstraites a déjà été détaillé. Nous ne présentons ici que l'utilisation qui en est faite par l'Editeur.

7.1.1. Les tables de structure et de présentation

Les tables de structure sont chargées et libérées dynamiquement par l'Editeur en fonction des besoins. Pour chaque nouveau document traité, l'Editeur charge d'abord la table de structure de sa classe puis les tables de structure de toutes les natures des objets qu'il contient (la liste de toutes ces natures se trouve en tête du fichier pivot). En cours de traitement, lorsqu'un objet est créé appartenant à une nature dont aucun représentant n'existe dans le document, la table de structure de cette nature est chargée. Lorsque le dernier objet d'une nature donnée est détruit, la table de structure de sa nature est libérée. Lorsque le traitement d'un document est terminé, sa table de structure est également libérée ainsi que celles de toutes les natures qu'il était seul à utiliser parmi les documents en cours de traitement.

Les tables de structure des documents ne sont pas partagées entre les documents. Si deux documents d'une même classe sont édités en même temps, l'Editeur utilise deux tables de structure. En fait ces deux tables ne sont pas strictement identiques, et c'est la raison pour laquelle elles ne sont pas partagées. Chaque table est complétée avec les règles qui référencent les tables de structure des natures créées dynamiquement grâce au type indéterminé UNIT. Les règles ajoutées sont supprimées de la table de structure lorsque le dernier élément qui y fait référence est détruit. Ainsi l'ensemble des règles ajoutées à la table de structure indique toutes les tables de structures nécessaires, en plus de celles qui sont explicitement référencées dans la table elle-même.

Puisque les tables de structure conservées en mémoire permanente décrivent des structures génériques de classes et non de documents particuliers, les règles ajoutées ne sont pas conservées dans les tables de structure après la session d'édition, mais elles sont conservées avec le document (voir la représentation pivot).

Les tables de présentation sont également chargées et libérées dynamiquement par l'Editeur, en fonction des besoins ; mais, à la différence des tables de structure, elles sont partagées entre les documents. Le partage est possible puisque ces tables ne sont pas modifiées pendant l'édition des documents.

7.1.2. Les images abstraites

On a déjà vu que l'Editeur maintenait plusieurs images abstraites simultanément, chacune représentant l'image d'une vue d'un document. Chaque image abstraite contient ce qui est effectivement affiché dans la fenêtre correspondante sur l'écran, et un peu plus pour que le Médiateur puisse réagir rapidement lorsque l'utilisateur fait avancer (ou reculer) de quelques lignes l'image affichée.

Pour plusieurs raisons il n'est pas envisageable de conserver l'image de la totalité de chaque document dans les images abstraites. Cela occuperait une place très importante en mémoire et induirait des calculs relativement lourds pour l'Editeur et le Médiateur à chaque modification du document, puisqu'un changement, même localisé, peut avoir des conséquences sur la présentation d'une grande partie du document (renumérotation de toute la suite, par exemple). En conservant l'image abstraite minimum, on réduit ces calculs et on utilise moins de mémoire. En contrepartie, à chaque déplacement dans le document, il faut reconstruire l'image abstraite de la nouvelle partie à afficher.

7.1.3. Les arbres abstraits

A la différence des images abstraites, les arbres abstraits, qui représentent la structure logique spécifique et le contenu des documents traités, sont complets en mémoire, c'est à dire qu'ils représentent la totalité de chaque document en cours d'édition.

Il est en effet nécessaire de disposer de la totalité de l'arbre abstrait pour le traitement de certaines commandes, comme les recherches ou les déplacements à travers le document. Plutôt que de maintenir une partie du document en mémoire et de gérer dans l'éditeur des échanges entre la mémoire et un fichier, nous avons préféré nous reposer sur un système opératoire utilisant une mémoire virtuelle, ce qui devient courant sur les postes de travail visés pour notre éditeur.

Ce choix est rendu possible par la relative compacité de la représentation abstraite. Celle-ci est unique pour chaque document : la notion de vue n'intervient qu'au niveau de la présentation, et il n'y a donc pas de copies multiples, en mémoire, de l'arbre abstrait d'un document. Cette représentation retient principalement la structure arborescente, les attributs et le contenu du document, ce qui augmente assez peu l'occupation mémoire par rapport à la simple représentation du contenu.

Chaque élément de la structure spécifique d'un document est représenté en mémoire par un bloc contenant :

- Le type de l'élément : un pointeur vers la table de structure de la classe à laquelle il appartient et le numéro dans cette table de la règle définissant sa structure. C'est la règle de structure qui contient le nom du type.
- Les pointeurs formant l'arbre : pointeurs vers l'élément père, l'élément précédent, l'élément suivant, et vers le premier fils si l'élément n'est pas un type de base (chaîne de caractère, élément graphique, symbole ou image).

- Les attributs affectés à l'élément, sous la forme d'un pointeur sur une chaîne de blocs, chaque bloc décrivant un attribut (triplet pointeur sur la table de structure définissant l'attribut, numéro de l'attribut dans cette table de structure, valeur de l'attribut). Si l'élément n'a pas d'attribut, le pointeur sur la chaîne des blocs attributs est simplement nul.

- Si l'élément est référencé, un pointeur vers le descripteur de référence (voir plus loin les références). Ce pointeur est nul si l'élément n'est pas référencé.

- Un ensemble de pointeurs sur les pavés représentant cet élément dans chacune des vues du document. Ces pointeurs sont nuls pour les éléments dont l'image abstraite n'est pas construite.

- Enfin, si l'élément est un type de base, le contenu est indiqué selon le type :
  - pour un symbole ou un élément graphique, un simple caractère (le code du symbole ou de l'élément graphique),
  - pour un texte ou une image, un pointeur vers une chaîne de caractères (pour une image, la chaîne contient le nom du fichier image),
  - pour une référence, un pointeur vers un bloc particulier décrivant la référence.

Un arbre abstrait est donc essentiellement constitué de tels blocs, chacun représentant un élément de la structure logique du document. On notera que dans le même arbre abstrait on peut trouver des éléments appartenant à des natures différentes, puisque chaque élément porte l'indication de la classe à laquelle il appartient (sous la forme d'un pointeur sur la table de structure). On a vu aussi que sur certains éléments viennent se greffer d'autres types de blocs :

- une chaîne de blocs attributs pour les éléments qui ont des attributs,

- une chaîne de caractères pour les feuilles de type texte ou image,

- deux types de descripteurs de référence, pour les éléments qui sont référencés et pour ceux qui référencent les premiers.

Ces deux derniers types de descripteurs ont été mis en place de façon à permettre une mise à jour rapide des références lors des modifications du document.

Un descripteur d'élément référencé est attaché à chaque élément auquel au moins une référence est faite dans le document. Il comporte :

- Un pointeur sur l'élément référencé lui-même.

- Un pointeur sur le premier descripteur de référence qui référence cet élément (tous les descripteurs de référence qui référencent le même élément forment une chaîne).
- Deux pointeurs qui assurent un chaînage dans les deux sens de tous les descripteurs d'éléments référencés du même document. Cette chaîne est ancrée dans le descripteur de document.

Un descripteur de référence est attaché à chaque élément du type référence qui apparaît dans un document. Il comporte :

- Un pointeur vers l'élément de type référence auquel il est attaché.
- Un pointeur vers le descripteur d'élément référencé de l'élément qu'il référence.
- Deux pointeurs qui assurent un chaînage dans les deux sens de tous les descripteurs de référence du même document qui référencent le même élément.

Cette double structure est illustrée par la figure 4.5 qui montre la représentation d'un extrait de document comportant deux références au même élément (la section 2) et deux références à deux autres éléments, des figures qui n'aparaissent pas dans l'arbre principal du document, puisque ce sont des éléments associés.

Grâce à tous ces chaînages, les mises à jour du document peuvent être très rapides. Par exemple, lorsqu'on ajoute une nouvelle section juste après l'introduction de notre exemple, l'ancienne Section 2 devient la Section 3. Cette section change de numéro, il faut donc changer les références qui pointent cette section et qui utilisent le numéro de section comme moyen de visualiser la référence. On voit dans l'arbre abstrait que l'ancienne deuxième section possède un descripteur d'élément référencé (pointeur symbolisé par le trait ----—). A partir de ce descripteur on trouve la chaîne des descripteurs de référence, et à partir de chacun de ces descripteurs, l'élément qui référence la section.

Tous les descripteurs d'éléments référencés du document sont chaînés de façon que l'on puisse les accéder rapidement, notamment lors de la sauvegarde du document, puisqu'on doit les étiqueter tous (voir la représentation pivot).

Un même document peut avoir plusieurs arbres abstraits en mémoire. Chaque document en a au moins un, celui qui décrit le document lui-même. Les autres arbres abstraits sont ceux qui représentent les différents ensembles d'éléments associés et les valeurs des paramètres. En effet les éléments associés ne font pas partie de l'arbre principal et n'y sont attachés que par le chaînage des références. Dans l'exemple de la figure 4.5, les deux figures qui apparaissent en bas sont des éléments associés reliés au document de cette façon.

1. INTRODUCTION

Xxx xxx xxxxx xxx xx xxxx dans la section 2 xx xxxxx xx xxxxx xx xxxxx xx xxx xxxxxx xx xxx xxxxx comme le montre la figure 1 xx xxxxx xxx xxxxx xx xxxxxx xx xxx xxxxx xxx xxxxxx.

Xx xxxxx xx xxxx xxx xxxxxx x xxxx xx xxxxxxxx xxx xxxxx xxx xxxxx xxx on verra dans la section 2 que xxx xxxxx xxx xxxxx xx x xxx xxxx xxx.

2. LES PRINCIPES

Xxxxxxx xx xxxx xxxxx xxx xxxxx xxx (voir figure 2). Xxxx xxxxx xxx xxx xxxxx xx...

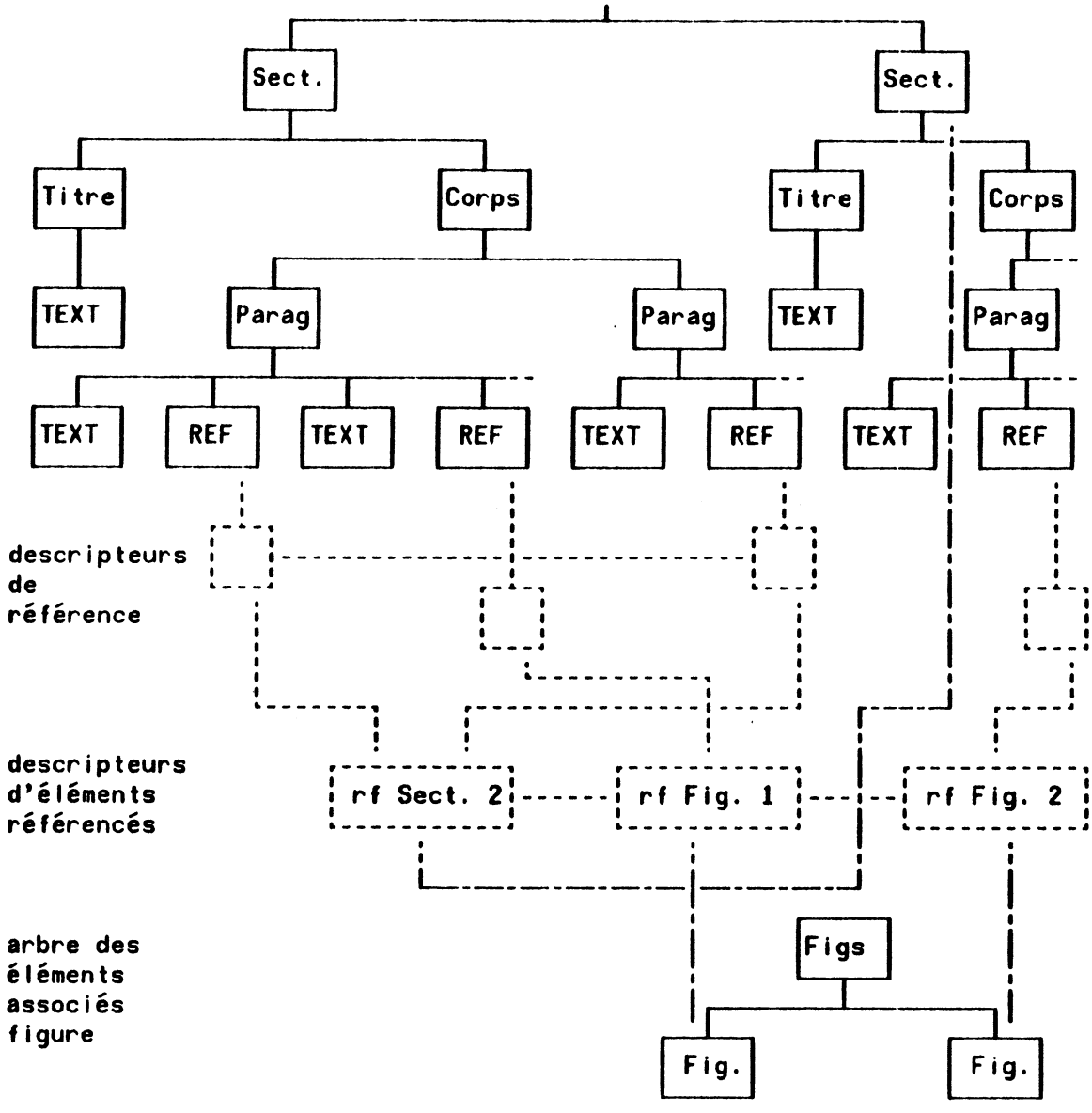


Figure 4.5 : Le chaînage des références.

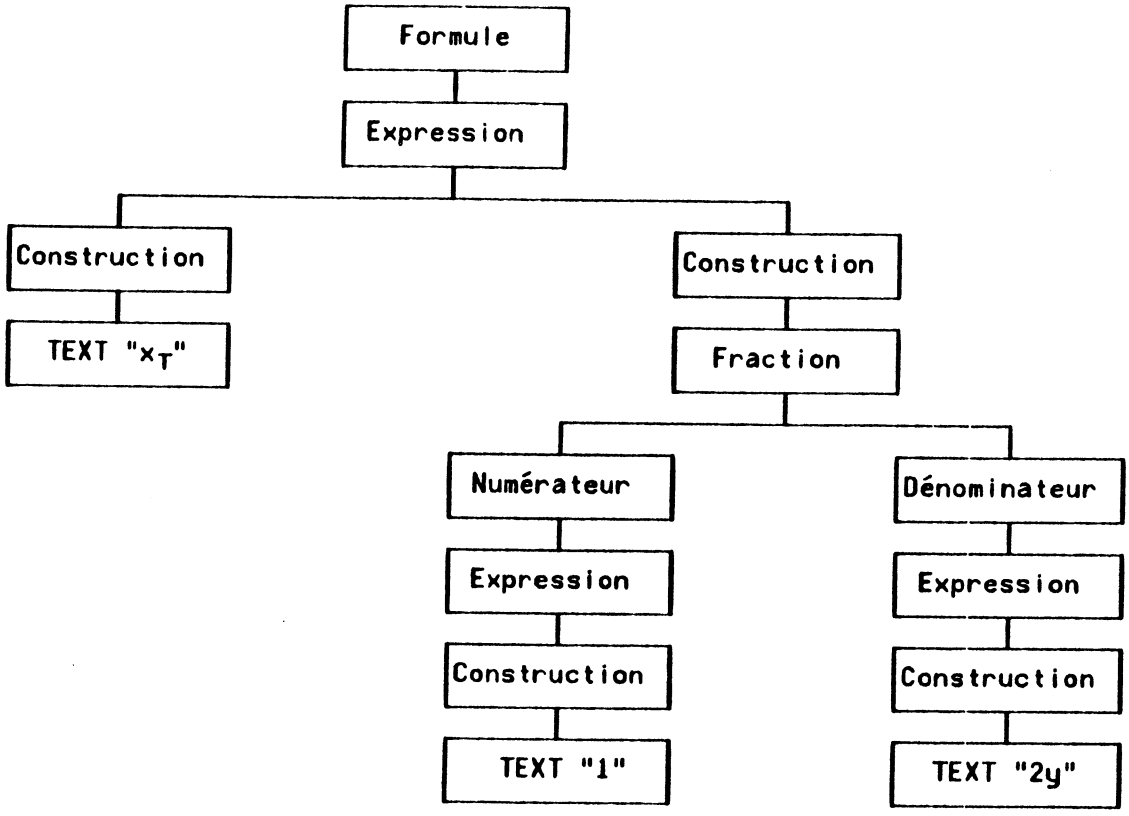
Pour permettre une construction rapide des vues, les éléments associés de même type d'un document sont regroupés sous la forme d'une liste. Cette liste constitue un arbre abstrait particulier, et un document possède autant de tels arbres qu'il a d'éléments associés de types différents. Ainsi les deux figures du bas de l'exemple précédent appartiennent au même arbre, celui qui regroupe toutes les figures du document. La racine de cet arbre est représentée par l'élément noté `Figs` sur le dessin.

Tous les arbres abstraits sont construits d'après les modèles constitués par les tables de structure. Une première version de Grif manipulait des arbres construits très directement d'après les schémas de structure. Ainsi, pour les formules mathématiques, le schéma suivant décrit la structure mise en œuvre dans Edimath :

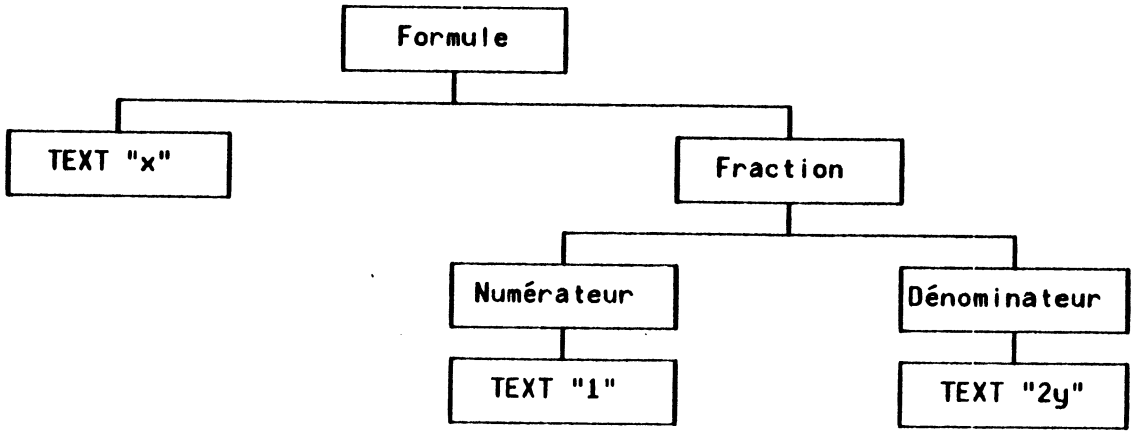
```
Formule =      Expression;
Expression =   LIST OF (Construction);
Construction = CASE OF
                 TEXT;
                 Fraction = BEGIN
                            Numérateur =   Expression;
                            Dénominateur = Expression;
                            END;
                 Racine = ...
                 ...
                 END;
```

Cette structure générique produisait un arbre très volumineux pour représenter même les formules les plus simples, comme celle-ci :

$$x = \frac{1}{2y}$$



Si cette structure est facile à construire en suivant simplement les règles de la table de structure, elle est très lourde à manipuler. Aussi, les règles de construction des arbres abstraits ont-elles été modifiées pour produire, à partir des mêmes tables de structure, des arbres plus simples, tel que celui-ci qui représente la même formule, construite d'aprés la même table de structure :



Cette simplification a été obtenue en ne créant que les nœuds

- qui sont les constituants d'un agrégat (c'est le cas de Numérateur et de Dénominateur),
- qui sont des éléments de liste (les Constructions auraient dû subsister, mais la dernière règle a été appliquée),
- qui sont l'une des options d'un choix (c'est le cas de Fraction),
- qui sont produits par la règle initiale d'un schéma de structure (c'est le cas de Formule),
- qui sont des éléments de base (c'est le cas des TEXT).
- Enfin, une option d'un choix se met à la place de ce choix, sauf s'il s'agit d'un élément d'agrégat : c'est ainsi que Fraction ou TEXT remplace Construction.

Grâce à ces règles de simplification, les structures construites ne contiennent que l'essentiel, et les éléments intermédiaires, qui ne sont introduits que pour rendre le schéma de structure plus clair, n'apparaissent pas dans les arbres abstraits. On obtient ainsi pour les formules mathématiques des arbres semblables à ceux qui sont créés par Edimath, si on prend un schéma de structure équivalent à celui d'Edimath. Pour les documents en général, le bénéfice est le même. Rappelons que l'utilisateur est prévenu, par le compilateur de schémas de structure, des types des éléments qui ne seront pas créés par suite de ces simplifications. Ainsi, il évite de définir des règles de présentation pour ces types.

7.2. Les manipulations de structure

7.2.1. La création des arbres abstraits

La création d'arbres ou de sous-arbres abstraits est effectuée de la même façon, qu'il s'agisse de la structure de l'ensemble d'un document (lorsque l'utilisateur crée un nouveau document), de la structure d'un objet (l'utilisateur crée l'objet dans un document) ou encore de n'importe quelle partie ajoutée à un document ou un à objet. L'Editeur suit la règle de structure du type de l'élément à créer et crée un élément de ce type, avec sa descendance, telle qu'elle est définie dans la table de structure. Il applique évidemment les règles de simplification que l'on vient de voir.

La descendance est créée de façon récursive : après avoir créé l'élément voulu, l'Editeur crée les descendants directs de cet élément, puis les descendants directs des éléments qu'il vient de créer, et ainsi de suite. Ce processus s'arrête lorsque l'Editeur ne

peut plus créer de descendants automatiquement. En effet, il ne crée pas de descendance pour :

- les éléments de base, comme les chaînes de caractères ou les éléments graphiques, qui constituent les feuilles des arbres abstraits.
- les éléments ayant pour constructeur choix : l'Editeur ne peut décider seul du type du descendant à créer ; l'utilisateur fera ce choix lorsqu'il créera le contenu de l'élément (voir plus loin les insertions).
- les éléments récursifs, marqués comme tels dans la table de structure par le compilateur de schémas de structure. On évite ainsi, par exemple, de créer une infinité de sections de niveaux inférieurs. S'il veut des sections du niveau immédiatement inférieur, l'utilisateur devra le demander explicitement, et un seul niveau sera créé à la fois.
- les références, qui n'ont jamais de descendance. C'est l'utilisateur qui désignera l'élément référencé.

Lorsqu'il crée la descendance d'un élément, l'Editeur suit le constructeur de cet élément. Il crée tous les éléments définis dans un agrégat. Il crée le nombre minimum d'éléments spécifié pour une liste (au moins un élément).

7.2.2. La sélection structurelle

Les modifications dans la structure d'un document ou d'un objet se font à partir de la sélection. La sélection opérée par l'Editeur est toujours faite de sorte que les commandes d'édition puissent opérer sur la partie couramment sélectionnée. Toutes les commandes d'édition acceptent le même type de structure de la partie sélectionnée.

Cette structure est essentiellement une suite de sous-arbres frères dans le même arbre abstrait. Le premier sous-arbre peut être précédé d'éléments de niveau inférieur ; de même, le dernier sous-arbre peut être suivi d'éléments de niveau inférieur. Les éléments de niveau inférieur en début ou en fin de sélection doivent être immédiatement voisins dans l'arbre, comme le montre l'exemple de la figure 4.6, où l'utilisateur a sélectionné la fin du texte d'un paragraphe, les deux paragraphes suivants en entier, et, dans le paragraphe suivant, le texte du début et la formule qui le suit. Si le premier ou le dernier élément de la sélection est une feuille de texte, elle peut être partiellement sélectionnée, mais la partie sélectionnée doit être contiguë au reste de la sélection : la fin de la feuille de texte dans le cas du premier élément de la sélection (comme sur la figure 4.6), ou le début dans le cas du dernier élément.

Cela représente la forme la plus générale que peut prendre la partie sélectionnée, mais le plus souvent, celle-ci se limite à un simple sous-arbre ou à quelques sous-arbres frères. Elle peut même se réduire à une simple feuille. Rappelons que lorsque la sélection concerne une partie d'une feuille (une sous-chaîne de caractères), elle n'est pas considérée par l'Editeur, mais par le Médiateur.

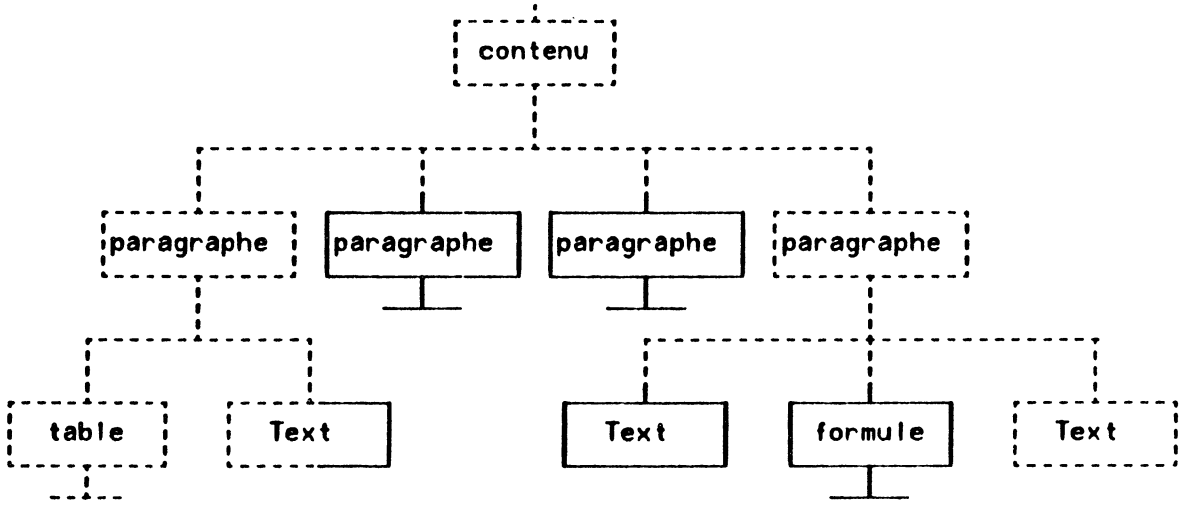


Figure 4.6 : Une sélection
(les éléments sélectionnés sont en trait plein).

Ce type de structure a été retenu pour que l'utilisateur puisse sélectionner tout ce qui se trouve sur l'écran, depuis un caractère jusqu'à un autre caractère, lui laissant ainsi toute liberté d'opérer comme il le ferait avec un système WYSIWYG. Il permet aussi d'effectuer une sélection par la structure, en ajoutant ou retranchant des éléments structurellement voisins.

Chaque fois que l'Editeur étend ou réduit la sélection courante jusqu'à un pavé désigné par l'utilisateur (ou plus exactement l'élément correspondant de l'arbre abstrait), la sélection courante est étendue ou réduite en respectant ces principes de structuration. Une fois la nouvelle sélection définie sur l'arbre abstrait, l'Editeur la visualise dans toutes les vues où elle est visible, en utilisant les services du Médiateur. L'utilisateur peut ainsi manipuler la sélection dans n'importe quelle vue et même changer de vue entre deux opérations successives d'extension ou de réduction de sélection.

A chaque changement de la sélection, l'Editeur met à jour l'ensemble des commandes définissables. Cette facilité offerte par le Médiateur est utilisée pour proposer à l'utilisateur à chaque instant tous les types d'éléments qu'il peut créer après la sélection courante, soit au niveau du dernier élément sélectionné, soit aux niveaux supérieurs. Cela facilite grandement la création d'un document, lorsque l'utilisateur procède de façon

séquentielle, créant les éléments dans l'ordre où ils resteront dans le document. Par exemple, avec le schéma de structure suivant :

```
Corps =       LIST OF (Chapitre);
Chapitre =    BEGIN
              Titre_Chapitre = TEXT;
              LIST OF (Section);
              END;
Section =     BEGIN
              Titre_Section = TEXT;
              LIST OF (Paragraphe);
              END;
Paragraphe =  LIST OF (Elément);
Elément =     CASE OF
                 TEXT;
                 Formule;
              END;
```

lorsque l'utilisateur entre du texte dans un paragraphe, les commandes qui lui sont proposées sont : `Elément`, `Paragraphe`, `Section`, `Chapitre`. Il peut ainsi, en frappant l'une des touches de fonction associées à ces commandes, créer immédiatement soit un nouvel Elément dans le même paragraphe (par exemple une formule), soit un paragraphe suivant dans la même section, soit une section suivante dans le même chapitre, soit encore un nouveau chapitre.

7.2.3. Le parcours du document

L'utilisateur peut parcourir le document qu'il édite en suivant sa structure ou son contenu. Pour cela il dispose d'une commande qui permet de rechercher, en avant ou en arrière, à partir de la sélection courante, différentes sortes d'informations.

La recherche la plus classique est celle des chaînes de caractères, que l'on trouve dans tous les éditeurs. Pour exécuter cette commande, l'Editeur parcourt l'arbre abstrait à la recherche des feuilles de texte successives et, dans chacune de ces feuilles il cherche la chaîne demandée. Dès qu'elle est trouvée, il la considère comme la sélection courante et demande au Médiateur de la mettre en évidence. L'utilisateur peut alors appliquer la commande de son choix sur la chaîne. Il peut notamment réitérer sa commande de recherche pour trouver la prochaine occurrence de la même chaîne de caractères.

De la même façon que pour une chaîne de caractères, l'utilisateur peut chercher un symbole donné, une image dont il spécifie le nom ou un élément graphique particulier.

La structure logique des documents est mise à profit pour les recherches de type et d'attribut. Ainsi, il est possible de rechercher un élément d'un type donné, ce type étant choisi parmi tous les types d'éléments définis pour le document concerné, c'est à dire les types définis dans le schéma de structure du document et dans les schémas de structure de tous les objets qu'il contient. De même, il est possible de chercher les éléments qui possèdent un attribut, soit un attribut quelconque, soit un attribut donné ayant une valeur quelconque, soit encore un attribut donné ayant une valeur donnée. Là aussi, tous les attributs définis pour le document sont utilisables dans la recherche.

Les références sont également considérées dans les commandes de recherche. L'utilisateur peut sélectionner un élément (avec la souris ou par une commande de recherche) et demander à l'Editeur de trouver la première référence à cet élément et les références successives au même élément (un élément peut être référencé plusieurs fois). Inversement, lorsqu'une reference est sélectionnée, l'Editeur peut chercher l'élément pointé par cette référence. Les structures de données mises en place pour les références facilitent ces recherches.

Les références sont prises en compte dans le parcours de l'arbre abstrait effectué dans les différentes commandes de recherche. Lorsque l'Editeur rencontre une référence dans ce parcours, et si cette référence pointe un élément associé, alors la recherche continue dans cet élément associé. Lorsque tout l'élément associé a été parcourru, la recherche reprend aprés la référence. Cela permet, dans toute recherche, de balayer l'ensemble du document, y compris les éléments associés.

Les commandes de recherche que l'on vient de voir utilisent la structure arborescente du document essentiellement pour définir un ordre entre les éléments lors du parcours. Elles n'utilisent pas la hiérarchie des arbres en tant que telle et lorsqu'on réitère une même commande de recherche on obtient des éléments dont les niveaux dans l'arbre abstrait diffèrent généralement. D'autres commandes permettent au contraire de se déplacer de façon hiérarchique dans le document. Le résultat de ces commandes, dites *commandes de sélection*, est le même que celui des commandes de recherche : l'élément trouvé devient la sélection courante ; mais le chemin suivi pour trouver l'élément est différent. Les commandes de sélection permettent de trouver, par rapport à la sélection courante, l'élément englobant (le père dans l'arbre abstrait), l'élément précédent ou suivant de même niveau (frères), le premier élément englobé (le premier fils). Si le traitement de ces commandes est trivial pour l'Editeur, leur intitulé peut paraître un peu abscons à un utilisateur peu familiarisé avec les structures arborescentes. Pour cette raison, l'Editeur construit dynamiquement le menu de sélection, où il indique le type des éléments environnant la sélection courante dans l'arbre abstrait. Ainsi l'utilisateur se voit proposer,

par exemple, un choix entre "Chapitre englobant", "Section précédente", "Section suivante", "Titre de section", lorsqu'une section est sélectionnée.

7.2.4. Les modifications des arbres abstraits

Les modifications des arbres abstraits se font en appliquant les quatre commandes d'édition (Insérer, Couper, Coller, Copier) à la partie sélectionnée, obtenue par l'un des moyens que l'on vient de voir.

Suppression

Lors de la suppression (commande Couper), très peu de contrôles sont effectués. La partie sélectionnée est simplement retirée de l'arbre abstrait. Cela peut conduire à un arbre qui ne respecte pas strictement le modèle du schéma de structure, puisqu'on peut ainsi obtenir des listes qui contiennent un nombre d'éléments inférieur au minimum spécifié dans le schéma. De même, un agrégat peut se voir retirer certains des éléments qui sont indiqués dans le schéma et qui sont automatiquement engendrés par l'Editeur lors de la création de l'agrégat. En fait cela correspond aux options qui ont été prises lors de la conception du modèle de document. Pendant l'édition, une certaine liberté est laissée à l'utilisateur vis à vis de la structure générique. Mais si une application requiert un strict respect de la structure générique, l'Editeur permet à l'utilisateur de compléter automatiquement le document de sorte qu'il soit conforme à la structure générique de sa classe (voir la commande "Insérer à l'intérieur").

La commande Couper conserve dans un tampon la partie retirée du document, mais uniquement sous la forme de l'arbre abstrait : les images abstraites, et donc concrètes, dans les différentes vues sont, elles, définitivement supprimées. Lorsqu'une commande Coller réutilisera cette partie, les images seront recalculées en fonction du nouveau contexte où se trouvera cette partie de document.

La commande Copier procède de la même façon que la commande Couper pour ce qui est de la conservation de la partie sélectionnée, mais elle ne modifie pas le document, sous aucune de ses formes : elle prend seulement une copie de la partie sélectionnée sous sa forme de l'arbre abstrait et la range dans le tampon.

Les références et les éléments référencés posent un problème particulier pour les commandes Couper et Copier. La solution retenue est la suivante : la commande Couper conserve dans le tampon les références qui se trouvent dans la partie sélectionnée, et leur descripteur de référence reste chaîné avec les autres descripteurs du document. De même, les descripteurs d'éléments référencés, pour les éléments référencés qui appartiennent à la sélection, restent dans la chaine des descripteurs du document. Ainsi, si le contenu du

tampon est ensuite utilisé par une commande Coller, on n'aura rien perdu, ni les références qui pointeront toujours les mêmes éléments, ni des éléments référencés qui, bien que déplacés, seront toujours pointés par les mêmes références.

La commande Copier procède différemment. Si certains éléments de la sélection sont référencés, la copie qui en est faite dans le tampon ne conserve pas le lien avec la référence. En effet, une référence ne peut pointer que sur un élément, et c'est l'élément original qui était dans le document, et non sa copie, qui est retenu. En revanche un même élément peut être référencé plusieurs fois. Les références figurant dans la sélection, avec leur descripteur de référence, sont donc copiées dans le tampon et les copies des descripteurs de référence sont chaînées avec les descripteurs de références des originaux. Ainsi, la prochaine commande Coller fera apparaître des références pointant sur les mêmes éléments.

Comme le modèle d'interaction retenu ne prévoit qu'un seul tampon pour conserver une partie de document, les deux commandes Couper et Copier procèdent à la destruction effective de l'ancien contenu du tampon avant d'y ranger la partie de document qu'elles doivent sauver. La destruction définitive de la représentation abstraite d'une partie sélectionnée a donc lieu, non pas sur la commande Couper, mais sur la prochaine commande Couper ou Copier.

Insertion

On a déjà vu, à propos de la sélection, que les commandes définissables offertes par le Médiateur permettent d'ajouter de nouveaux éléments dans un document. Mais elles ne permettent que d'ajouter à la suite de l'élément courant. Or il est souvent nécessaire d'ajouter avant ou à l'intérieur d'un élément sélectionné. Cela est possible grâce à deux commandes prédéfinies :

- la commande Insérer qui crée de nouveaux éléments,
- la commande Coller qui récupère les éléments qui ont été rangés dans le tampon par la dernière commande Couper ou Copier.

Avec ces deux commandes l'utilisateur peut choisir la position où se placeront les éléments ajoutés : devant la partie sélectionnée, après elle ou encore à l'intérieur. Ce choix lui est offert à travers un menu qui est activé par l'Editeur dès que l'une de ces deux commandes est appelée.

Les créations provoquées par les commandes définissables sont effectuées de la même façon que celles qui sont déclenchées par la commande Insérer.

Insérer à l'intérieur

La création à l'intérieur d'un élément est d'abord faite pour compléter une création précédente qui s'était arrêtée sur un choix ou un élément récursif, mais aussi pour recréer dans un élément le contenu spécifié par le schéma de structure et dont une partie a été détruite par la commande Couper. L'Editeur procède différemment selon le type de l'élément dont il crée le contenu :

- Si l'élément sélectionné est un choix, l'Editeur compose, d'après le schéma de structure, un menu avec tous les types possibles pour ce choix et il demande au Médiateur de présenter le menu à l'utilisateur. Celui-ci n'a qu'à indiquer le type qu'il souhaite et l'Editeur crée tout le sous-arbre correspondant, qu'il vient mettre à la place de l'élément choix dans l'arbre abstrait.

- Si l'élément sélectionné est une référence, l'Editeur demande, à l'utilisateur, par l'intermédiaire du Médiateur, de désigner sur l'écran l'élément qu'il veut référencer. Les références étant typées dans les schémas de structure, l'Editeur cherche alors le premier élément englobant l'élément désigné et qui est du type prévu par le schéma. Il établit alors les chaînages du descripteur de référence et du descripteur de l'élément référencé.

- Si l'élément sélectionné est un élément récursif, il crée sa descendance comme on l'a vu, en s'arrêtant après la création du premier élément de même type.

- Si l'élément sélectionné est un agrégat, il crée, avec leur descendance, tous les composants de l'agrégat prévus par le schéma de structure et qui sont absents.

- Si l'élément sélectionné est une liste, il ajoute des éléments de façon à atteindre le nombre minimum d'éléments spécifié dans le schéma.

Insérer Devant, Insérer Derrière

La création avant ou après la partie sélectionnée permet d'ajouter un élément dans une liste ou de recréécr un composant d'agrégat qui a été détruit par une commande Couper.

- Si la sélection courante ne porte pas sur un élément d'agrégat ou de liste, la commande ne peut pas s'exécuter. Pour éviter un refus brutal, l'Editeur essaie d'appliquer la commande à un autre niveau dans l'arbre. Il tente d'abord d'insérer devant (respectivement derrière) les éléments de niveau supérieur, si l'élément n'a pas de prédécesseur (respectivement de successeur) à son niveau. En cas d'échec, il essaie d'insérer à l'intérieur, au début (respectivement à la fin) du premier élément suivant (respectivement précédent). Cela permet à l'utilisateur de sélectionner ce

qu'il voit sur l'écran, sans prêter trop d'attention au niveau structurel : l'Editeur trouve lui-même le niveau dans l'arbre abstrait auquel l'opération peut avoir lieu.

- Si le premier (respectivement le dernier) élément de la sélection courante est un élément de liste, un nouvel élément de même type, avec toute sa descendance, est créé devant (respectivement derrière) la sélection. Toutefois, si le nombre maximum d'éléments spécifié dans le schéma de structure est déjà atteint, la commande est sans effet.

- Si le premier (respectivement le dernier) élément de la sélection courante est un élément d'agrégat, un élément voisin du type prévu dans le schéma est créé, avec toute sa descendance, devant (respectivement derrière) la sélection. Toutefois, si l'élément existe déjà dans le document, la création est refusée.

Coller

La commande Coller vérifie d'abord si le contenu du tampon peut, d'après le schéma de structure, être inséré à l'endroit voulu par l'utilisateur. Au besoin, l'Editeur adapte la structure de la partie à coller, soit en ignorant l'élément de plus haut niveau du tampon et en prenant son contenu, soit, au contraire, en créant des éléments intermédiaires entre l'élément ou l'on colle et les éléments à coller.

Ce type d'adaptation simplifie beaucoup l'utilisation des commandes Copier, Couper, Coller. En effet, sans cette facilité, il faudrait que l'utilisateur détermine précisément le niveau de la partie à copier ou à couper en fonction de la structure où il veut la coller ultérieurement. Prenons, par exemple, le schéma de structure suivant :

```
Section = BEGIN
          Titre_Section = TEXT;
          Corps_Section = LIST OF (Paragraphe);
          END;
```

Si, dans un document, le corps de la section 1 contient trois (et seulement trois) paragraphes que l'on souhaite dupliquer après le dernier paragraphe du corps de la section 2, il faudrait, avant d'appeler la commande Copier, sélectionner les trois paragraphes, et non pas le corps de la section 1, qui occupe pourtant le même espace sur l'écran. En effet, un contrôle strict des structures provoquerait un refus de l'Editeur à l'appel de la commande Coller, puisqu'un corps de section ne peut pas suivre un paragraphe. Au contraire, avec l'adaptation qui est faite, l'utilisateur peut sélectionner indifféremment les trois paragraphes ou le corps de la section 1, puisque dans le deuxième cas l'Editeur ignorera le corps de section qui est dans le tampon et ne collera que son contenu, les trois paragraphes.

Inversement, si on ne sélectionne que deux paragraphes de la section 1 avant de Copier et qu'on veut les Coller dans une section qui n'a pas de corps (supprimé auparavant par une commande Couper), un contrôle strict des structures imposerait de créer d'abord un corps de section avant de pouvoir Coller. Grâce aux adaptations de structure, la commande Coller peut être appelée immédiatement, puisqu'elle créera elle-même le corps de section nécessaire avant d'y mettre les deux paragraphes contenus dans le tampon.

Lors de l'exécution de la commande Coller, les références posent deux problèmes distincts. L'un concerne les éléments à coller qui sont référencés depuis différents points du document, l'autre concerne les références qui doivent être collées. Le premier problème a été résolu de la façon suivante : si, à la suite d'une commande Couper, le tampon contient des éléments référencés, la commande Coller modifie les références à ces éléments de façon qu'elles pointent sur les copies des éléments qui sont insérées dans le document par l'exécution de la commande. Les éléments du tampon ne sont donc plus référencés après la première commande Coller. Cette solution a été retenue pour faciliter le déplacement (Couper puis Coller) d'une partie de document tout en mettant à jour les références qui pointent vers cette partie.

Le deuxième problème a été traité différemment. Les références qui se trouvent dans le tampon ne sont pas modifiées. Chaque commande Coller les duplique dans la partie insérée dans le document, les copies des références pointant sur les mêmes éléments que les références qui sont dans le tampon. Cependant, si l'opération Coller a lieu dans un document qui n'est pas celui où le contenu du tampon a été prélevé, les références ne sont pas conservées. Plus exactement, des éléments références sont bien introduits dans le document, mais ils sont vides : ils ne pointent aucun élément. L'utilisateur est alors libre de les détruire (Couper) ou de désigner l'élément qu'ils doivent pointer (Insérer à l'intérieur) dans le nouveau document.

7.3. L'affichage et le réaffichage

Conformément à l'architecture qui a été retenue, l'affichage et le réaffichage des vues des documents en cours d'édition sont effectués par le Médiateur, à partir des indications fournies par l'Editeur dans les images abstraites.

Création

Lors de la création d'une nouvelle vue, ou à la suite d'un déplacement à travers le document, une nouvelle image abstraite est créée par l'Editeur en parcourant l'arbre abstrait de la partie de document à afficher et en appliquant à chaque nœud de cet arbre les règles de présentation associées au type de l'élément nœud. Lorsqu'un paramètre de présentation n'est pas défini explicitement par une règle associée au type de l'élément,

l'Editeur applique la règle de présentation définie par défaut pour ce paramètre dans le schéma de présentation. Lorsque toute l'image abstraite de la partie affichable du document a été construite, l'Editeur la fait visualiser par le Médiateur.

Dès qu'une modification a été effectuée sur un arbre abstrait par l'une des commandes d'édition, cette modification est répercutée sur toutes les images abstraites correspondant aux vues du document modifié. Dans chacune de ces images, l'Editeur évalue tous les pavés qui sont affectés. Pour cela, il traite de façon différente les ajouts de nouveaux éléments et les suppressions.

Ajouts

Pour ajouter de nouveaux pavés dans l'arbre d'une image abstraite, l'Editeur crée d'abord les pavés de tous les nouveaux éléments, comme il le fait lors de la création d'une nouvelle image. Puis il examine les conséquences qu'a sur les pavés voisins l'ajout de la racine du nouveau sous-arbre, et seulement de la racine, puisque les règles de présentation n'introduisent des relations qu'entre pavés frères ou entre pavé père et pavé fils. La présentation de tout le sous-arbre ajouté peut donc être calculée indépendamment du contexte, et seule la racine affecte les pavés environnants.

L'Editeur vérifie donc dans le schéma de présentation si les règles de positionnement des axes de reférence du pavé englobant et ses règles de dimensionnement se réfèrent au nouveau pavé. Si c'est le cas, il réapplique ces règles pour le pavé englobant. Il procède de même pour les règles de positionnement et de dimensionnement des pavés frères du nouveau pavé. Tous les pavés dont au moins une règle a été réappliquée sont marqués comme changés et les pavés nouvellement créés sont marqués comme nouveaux. L'Editeur demande enfin au Médiateur de réafficher toutes les vues où au moins un changement a eu lieu. Grâce aux marques, le Médiateur connaît les pavés à traiter. Il les traite donc puis supprime les marques.

Suppression

Lors d'une suppression d'éléments dans un arbre abstrait, l'Editeur marque d'abord comme morts tous les pavés correspondant à ces éléments, puis il cherche dans le pavé englobant et dans les pavés frères de la racine de chacun des sous-arbres supprimés les paramètres de présentation (axe de référence, dimension et position) qui se réfèrent au pavé racine supprimé. Les règles définissant ces paramètres dans le schéma de présentation sont réappliquées en faisant abstraction des pavés morts, de sorte que les pavés environnants se positionnent et se dimensionnent uniquement par rapport aux pavés qui restent. Puis les pavés pour lesquels les règles ont été réappliquées sont marqués changés. Le Médiateur est alors appelé pour le réaffichage, et il met à jour l'image sur l'écran. C'est seulement après l'affichage que l'Editeur supprime de l'image abstraite tous les pavés marqués morts.

Mais les ajouts et suppressions d'éléments n'ont pas seulement une influence sur les éléments immédiatement environnants : pavés père et frères ; ils peuvent avoir des conséquences sur les éléments de numérotation et les références qui se trouvent plus loin, avant ou après la partie modifiée. Par exemple, si on ajoute ou si on supprime une formule numérotée dans un document, les formules suivantes doivent changer de numéro et les références à ces formules doivent également changer si elles affichent le numéro de la formule référencée.

Numérotation

Pour effectuer la renumérotation, l'Editeur cherche d'abord parmi tous les pavés créés ou à détruire (et pas seulement dans les racines des sous-arbres créés ou détruits, mais à l'intérieur de ces sous-arbres) ceux qui affectent les compteurs. Cette recherche est facilitée par le fait que les tables de présentation associent à chaque compteur la liste des types d'éléments qui affectent la valeur du compteur. Pour chaque compteur affecté l'Editeur cherche alors, dans l'image abstraite, les pavés de présentation qui font apparaître la valeur de ce compteur et qui se trouvent après la partie ajoutée ou supprimée. Là encore, la table de présentation facilite la tâche de l'Editeur, puisqu'elle indique pour chaque compteur la liste des types de pavés de présentation qui utilisent sa valeur. Pour chacun de ces pavés de présentation, la valeur du compteur est alors recalculée sur l'arbre abstrait et le pavé de présentation est mis à jour puis réaffiché par le Médiateur.

Ce processus de renumérotation peut paraître lourd, puisqu'il faut chercher tous les pavés qui affectent les compteurs et tous les pavés qui affichent la valeur des compteurs affectés. Mais ces recherches ne se font que sur la partie affichée du document, celle pour laquelle il existe des images abstraites, c'est à dire une petite partie d'un document (environ une page ou un écran) dont la taille est indépendante de celle du document. La renumérotation est donc toujours rapide, puisqu'elle ne recalcule que les numéros visibles. Les autres numéros ne seront recalculés, à partir de l'arbre abstrait, que lorsqu'ils seront affichés.

Références

Une fois les numéros visibles recalculés, il faut réafficher toutes les références qui copient ces numéros. Ainsi les notes (de bas de page pour un document imprimé) sont numérotées et les renvois à ces notes, qui sont des références, sont couramment visualisés par ces numéros. Donc, chaque fois qu'un pavé de présentation qui contient un numéro est modifié, l'Editeur cherche dans l'arbre abstrait toutes les références qui sont faites à l'élément auquel correspond le pavé de numérotation (les chaînages de l'arbre abstrait permettent de trouver rapidement toutes ces références). Si ces références sont visibles (elles possèdent au moins un pavé) et si elles affichent, par une règle COPY, le numéro qui a changé, la règle COPY leur est réappliquée et leur pavé est réaffiché.

Comme pour les numéros, les recherches des références à réafficher sont rapides, puisqu'elles ne portent que sur la partie affichée du document : on ne cherche que les références qui pointent un élément affiché, et parmi celles-ci que celles qui sont elles-mêmes affichées. Le délai de calcul de ces références est donc, lui aussi, indépendant du volume du document.

Le seul inconvénient, mineur au demeurant, est que certaines références visibles peuvent pointer vers des éléments non visibles. Comme les références sont souvent visualisées par copie du numéro de l'élément référencé, et que les numéros ne sont calculés que lorsque les éléments sont affichés, certaines références ne peuvent pas être visualisées normalement. L'Editeur représente de telles références sous la forme d'une étoile entre crochets[*], mais dès que l'élément référencé est affiché, il remplace cette forme par la forme normale. Ainsi, dans un document utilisant des éléments associés notes, tant que la vue des notes n'est pas créée tous les renvois aux notes sont figurés par une étoile entre crochets. Dès que l'utilisateur crée la vue des notes, celles-ci sont numérotées par l'Editeur et ces numéros sont reportés dans les renvois. Seuls les renvois à des notes non affichées (fenêtre des notes trop petite, notes trop nombreuses...) restent sous la forme indéterminée d'une étoile. Cela ne gêne pas vraiment l'utilisateur puisque de toutes façons il ne peut pas voir la note référencée.

Il faut distinguer entre une référence qui pointe sur un élément non affiché et une référence qui ne pointe sur aucun élément. L'Editeur les différencie en affichant les premières sous la forme [*] et les secondes sous la forme [?]. Rappelons que la commande "Insérer à l'intérieur" permet pour les secondes de désigner l'élément qu'elles doivent pointer.

7.4. Les éléments hors structure

Nous avons jusque là présenté essentiellement l'arbre abstrait principal du document, celui qui décrit la structure spécifique du document proprement dit. Mais notre modèle de document prévoit également des éléments associés et des paramètres, qui appartiennent au document sans faire partie de l'arbre principal.

7.4.1. Les éléments associés

Les éléments associés sont regroupés selon leur type, comme on l'a vu sur la figure 4.5. Pour chaque type d'élément associé l'Editeur crée un arbre abstrait, dont la racine représente l'ensemble des éléments de ce type appartenant au document, les éléments associés eux-êmes étant les fils de cet élément. Ils forment ainsi une liste ordonnée. Lors de la création d'un nouveau document, l'Editeur crée un tel élément racine, sans descen-

dance, pour chacun des types d'éléments associés déclarés dans le schéma de structure. Rapelons que le compilateur de schémas de structure a préparé ce travail en ajoutant dans la table de structure une règle Liste pour chaque type d'élément associé. Grâce à cette règle Liste, l'utilisateur pourra ajouter simplement de nouveaux éléments associés, avec les commandes habituelles Insérer ou Coller.

Chaque fois que l'utilisateur crée une vue pour des éléments associés, l'Editeur y fait afficher l'image abstraite de l'arbre abstrait des éléments associés de ce type. A l'intérieur d'une de ces vues, l'édition se fait exactement comme dans n'importe quelle autre vue.

7.4.2. Les paramètres

Les paramètres constituent également des arbres abstraits séparés, avec un arbre abstrait pour chaque paramètre dont la valeur est définie. L'éditeur ne permet pas de modifier les valeurs des paramètres, puisque cette opération est réservée à certaines applications particulières. Ce sont ces applications qui, travaillant sur les fichiers contenant la représentation pivot des documents, prélèvent certaines parties d'un document pour les insérer en tant que paramètres dans d'autres documents.

Prenons l'exemple d'une application de publi-postage. Cette application travaille sur deux types de documents : des listes d'adresses et des lettres. Ces deux classes de documents peuvent être définies par les schémas de structure suivants :

```
STRUCTURE Liste_adresses;
STRUCT
   Liste_adresses = LIST OF (Adresse);
   Adresse =        BEGIN
                    Nom = TEXT;
                    Adresse_Postale = LIST OF (Ligne_adresse = TEXT);
                    Ville = TEXT;
                    END;
END

STRUCTURE Lettre;
PARAM
   Par_Nom_Dest = TEXT;
   Par_Adresse = LIST OF (Ligne_adresse = TEXT);
   Par_Ville = TEXT;
STRUCT
   Lettre = BEGIN
            Adresse_dest = BEGIN
```

```
                                Par_Nom_Dest;
                                Par_Adresse;
                                END;
                    Corps = LIST OF (Paragraphe);
                    Signature = TEXT;
                    END;
        Paragraphe = LIST OF (Elément = CASE OF
                                            TEXT;
                                            Par_Nom_Dest;
                                            Par_Ville;
                                            END;
                                );
END
```

Grif est utilisé pour constituer d'une part des lettres types et d'autre part des listes d'adresses. Lors de la création d'une lettre type, Grif crée en tête, conformément au schéma de structure des lettres, une adresse avec les deux paramètres et il laisse l'utilisateur insérer dans les paragraphes, là où il le souhaite, le nom du destinataire ou le nom de sa ville. Mais comme il s'agit de paramètres, l'utilisateur ne pourra pas en spécifier le contenu et ces éléments seront visualisés sous la forme de leur nom, encadré de deux caractères '$'. Dans les listes d'adresse ces mêmes éléments ne sont pas déclarés comme paramètres et leur contenu peut donc être entré et modifié librement.

L'application de publipostage, travaillant à partir d'un document lettre type et d'un document liste d'adresses, crée autant de lettres personalisées qu'il y a d'éléments Adresse dans le document liste d'adresses. Pour cela, elle crée une copie de la lettre type et ajoute en tête, en tant que paramètres (voir plus loin le format des fichiers pivot), les trois éléments constituant une adresse dans le document liste d'adresses. Elle ne modifie ni le contenu ni la structure de la lettre, elle ajoute simplement les paramètres en tête. Les documents ainsi produits, s'ils sont traités par l'éditeur Grif, présentent aux endroits indiqués dans la lettre type les *valeurs* des paramètres, mais l'Editeur interdit à l'utilisateur de modifier ces valeurs.

Lorsque l'Editeur lit un document depuis un fichier pivot, il forme un arbre abstrait pour chaque paramètre dont la valeur apparaît en tête du fichier. Ces arbres ne sont jamais modifiés par l'Editeur, qui ne crée d'ailleurs pour eux aucune image. Lorsqu'un paramètre apparaît dans le document, ou lorsqu'il est créé par l'utilisateur, l'Editeur vérifie s'il possède un arbre abstrait représentant sa valeur. Si le paramètre n'a pas de valeur pour le document, un simple élément de base Chaîne de caractères (contenant le nom du paramètre entre deux '$') représente alors le paramètre. Si le paramètre a une valeur, l'arbre abstrait représentant la valeur est copié pour fournir la représentation de l'élément.

Que le paramètre ait une valeur ou non, dans l'image abstraite toutes les feuilles qui en font partie portent l'indication "non modifiable" de sorte que le Médiateur ne permet pas de changer leur contenu. De même, pour l'Editeur, les éléments structurés qui forment les copies de la valeur du paramètre ne sont pas modifiables, sauf la racine du sous-arbre, qui représente la totalité du paramètre, puisqu'on peut ajouter ou supprimer un paramètre, mais pas modifier sa valeur.

7.5. La représentation pivot

Les documents traités par Grif sont conservés dans des fichiers, sous la forme de leur représentation pivot. Cette représentation décrit le contenu du document ainsi que sa structure spécifique, mais elle ne contient aucune information concernant la présentation.

Les principaux critères qui ont présidé à la définition de la représentation pivot sont la compacité de cette représentation et l'efficacité des traitements qui lui sont appliqués. La compacité est nécessaire pour obtenir des fichiers documents de faible volume et l'efficacité doit permettre d'effectuer rapidement les opérations de chargement d'un document en mémoire (internalisation) et de rangement d'un document dans un fichier (externalisation).

Pour obtenir une bonne efficacité, la forme pivot est constituée d'une représentation préfixée des arbres abstraits du document. Les contenus des feuilles des arbres apparaissent à l'intérieur de cette représentation préfixée, de sorte que l'internalisation et l'externalisation des documents se font par un simple accès séquentiel du fichier.

Pour une plus grande compacité, tous les types, attributs, constantes, etc..., qui sont définis dans les tables de structure, sont indiqués dans la représentation pivot sous la forme d'un nombre qui est le rang de l'entrée de la table qui les définit. Les tables de structure sont donc nécessaires à la compréhension de la forme pivot d'un document, mais comme elles sont partagées par tous les documents ou les objets d'une classe, elles ne sont pas répétées dans chaque fichier document, qui comporte seulement en tête la liste des structures génériques utilisées dans le document.

A la différence des autres représentations de documents qui sont manipulées par l'Editeur, la représentation pivot est complètement linéaire. Elle représente bien des structures arborescentes, mais celles-ci sont mises à plat. Cela permet notamment de stocker des documents dans des fichiers, mais aussi de les transmettre vers d'autres machines, sur des supports de transmission série.

Si la forme préfixée permet effectivement de mettre un arbre à plat, il reste le problème des références, qui ne sont pas des relations arborescentes. Ce problème a été résolu en étiquetant, au moment de l'externalisation, tous les éléments référencés du document. Cela se fait très simplement puisque les descripteurs des éléments référencés d'un document sont tous chaînés (voir figure 4.5). L'étiquette est le numéro d'ordre du descripteur dans cette chaîne. Ainsi chaque élément référencé est muni d'une étiquette unique, qui est mise en tête de l'élément dans la forme pivot. Dans cette même forme pivot, les références portent l'étiquette de l'élément référencé. En affectant les étiquettes avant de construire la forme pivot on peut traiter aussi bien les références en avant que les références en arrière.

Dans les arbres abstraits, des attributs sont associés à certains éléments individuellement. Les valeurs des attributs s'appliquent à l'élément auquel ils sont associés ainsi qu'à tout son sous-arbre. Si dans le sous-arbre un élément porte le même attribut avec une valeur différente, cet élément et tout son sous-arbre prennent cette valeur de l'attribut. En d'autres termes, pour chaque attribut, chaque élément hérite la valeur associée à lui-même ou à défaut au plus proche de ses ascendants. Dans la représentation pivot, les attributs sont indiqués conjointement aux éléments auxquels ils sont associés.

Pour plus de détails, on se reportera à l'annexe 4 qui donne une description détaillée de la syntaxe de la représentation pivot.

8. QUELQUES EXEMPLES

Les quelques exemples qui suivent montrent l'aspect graphique des documents affichés sur l'écran et donnent une indication de l'interface utilisateur : les commandes et menus disponibles et leur enchaînement pour réaliser quelques opérations d'édition courantes.

8.1. Différents types de documents

La figure 4.7 présente l'édition d'un document bilingue. Pour ce type de document, le schéma de présentation choisi permet la création de deux vues en plus de la vue principale : l'une affiche la partie française du document, l'autre la partie anglaise. Seule la vue française a été créée par l'utilisateur. On remarquera, en haut de chaque vue, les boîtes de présentation qui indiquent la date du jour, avec des formats différents en français et en anglais.

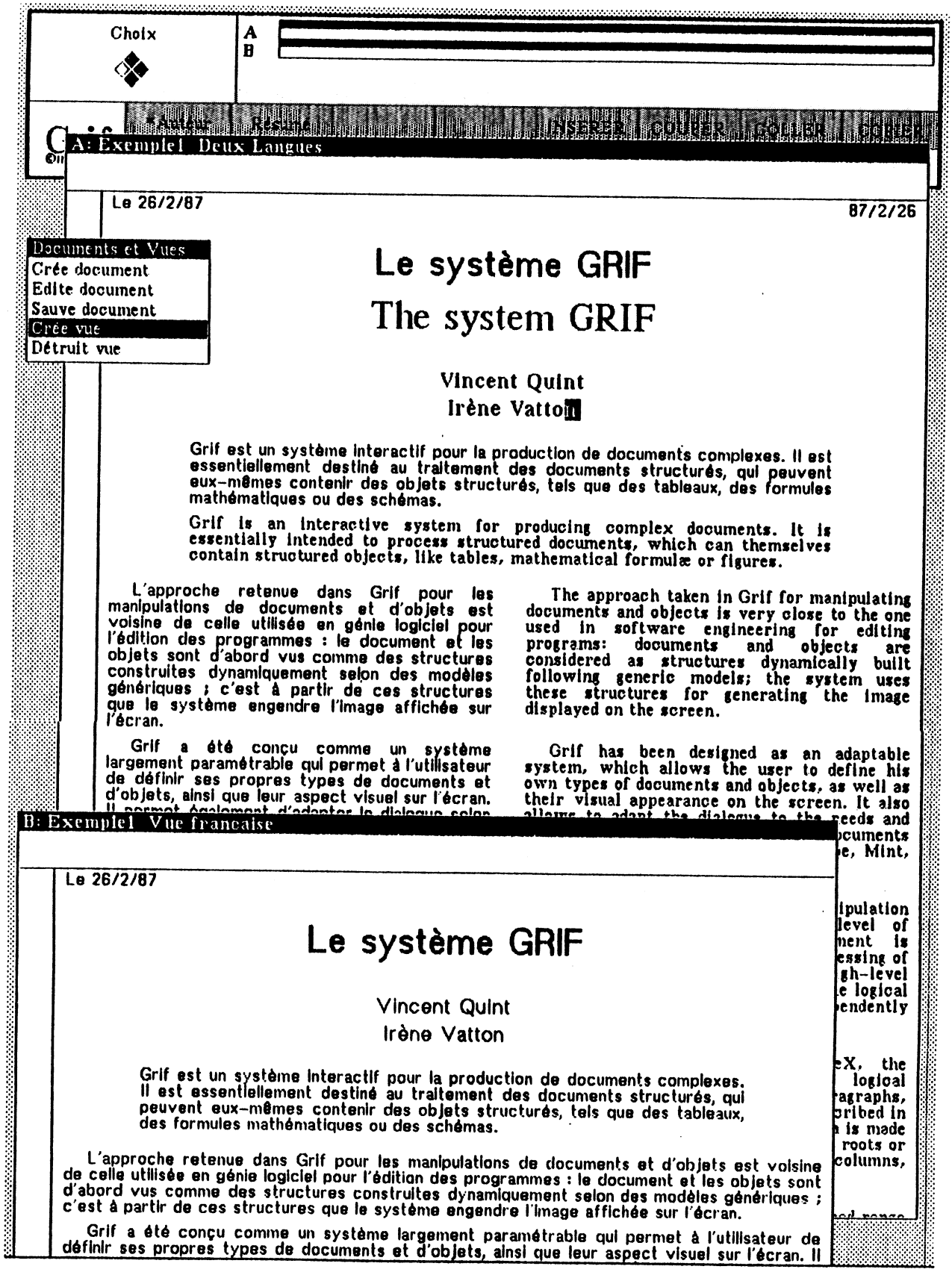


Figure 4.7 : L'image d'un document bilingue.

Le menu affiché en haut à gauche de l'écran donne la liste des commandes qui s'appliquent aux documents et aux vues. Ici, l'utilisateur demande à créer une nouvelle vue. Le système va donc présenter un nouveau menu, contenant toutes les vues qu'il est encore possible de créer. Pour ce document, seule la vue anglaise peut encore être créée.

Les huit cases grises en haut de l'écran correspondent aux huit touches de fonction du clavier. Les quatre de droite sont les commandes permanentes, les quatre de gauche sont les commandes de création et de déplacement, qui dépendent du contexte de la sélection courante. Le curseur se trouve à la fin du nom du dernier auteur ; les touches de fonction permettent donc d'ajouter un nouvel auteur ou de passer au résumé. Lorsque le nom d'un type d'élément est précédé d'une étoile, la commande crée un élément de ce type, après la position courante, sinon elle fait simplement passer au prochain élément du type indiqué.

La figure 4.8 présente l'édition d'un formulaire. Ici, les deux fenêtres correspondent à deux documents différents de la même classe. La fenêtre partiellement masquée montre un document vide, tel qu'il apparaît lorsque l'utilisateur crée un nouveau document de la classe "Formulaire mission". Les filets et les différents textes visibles dans cette fenêtre sont des boîtes de présentation qui ont été créées par l'éditeur, selon le schéma de présentation retenu. Les rectangles gris représentent des éléments vides, les champs à remplir par l'utilisateur. La deuxième fenêtre affiche un autre formulaire, qui, lui, a été complètement rempli.

Les rectangles gris représentant les éléments vides ne sont pas toujours explicites. Dans le cas de ces formulaires, les boîtes de présentation lèvent toute ambiguïté, mais lorsque ce n'est pas le cas, il suffit de désigner, avec la souris, un élément (vide ou plein), pour que son type s'affiche dans la zone des messages, en haut de l'écran. Ici, l'élément sélectionné, affiché sur fond noir, est la destination.

Le menu présenté en haut à droite est le menu de sélection dont le contenu varie avec la sélection courante. Il permet de sélectionner l'élément englobant (commande ^), l'élément précédent (commande <), l'élément suivant (commande >) et le premier élément englobé (commande _). Chacune de ces commandes indique le type de l'élément concerné.

La figure 4,9 montre trois documents mis en jeu dans une application de publi-postage. La lettre de la fenêtre C est la lettre type. Elle contient différents paramètres, qui n'ont pas de valeur et qui sont donc visualisés par leur nom entre deux caractères '$'. La fenêtre A affiche le document liste d'adresses qui fournit à l'application de publi-postage les valeurs des différents paramètres. Enfin la troisième fenêtre affiche une lettre construite par l'application qui a affecté aux paramètres de la lettre type les valeurs contenues dans l'une des adresses de la liste.

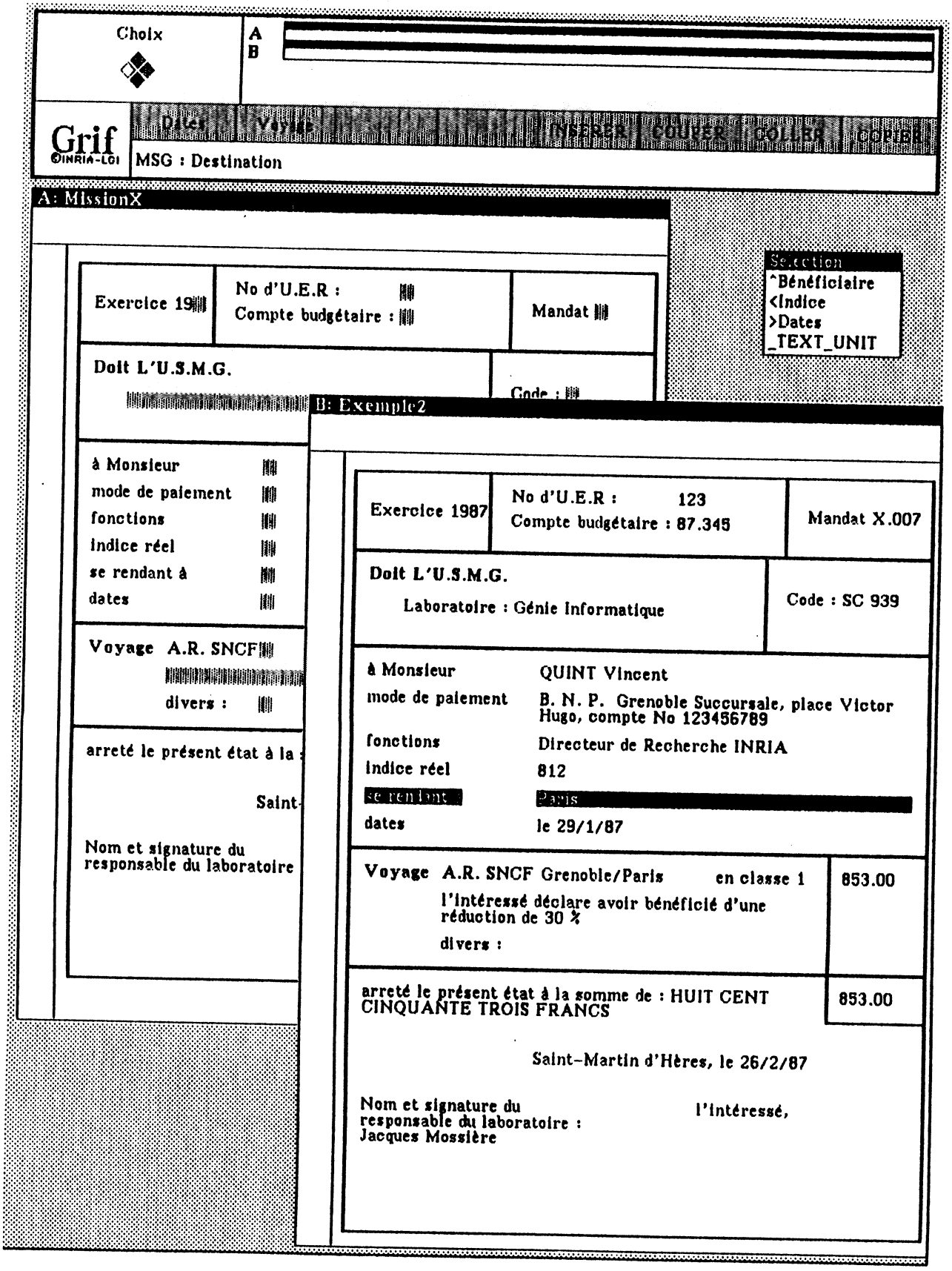


Figure 4.8 : L'image d'un formulaire.

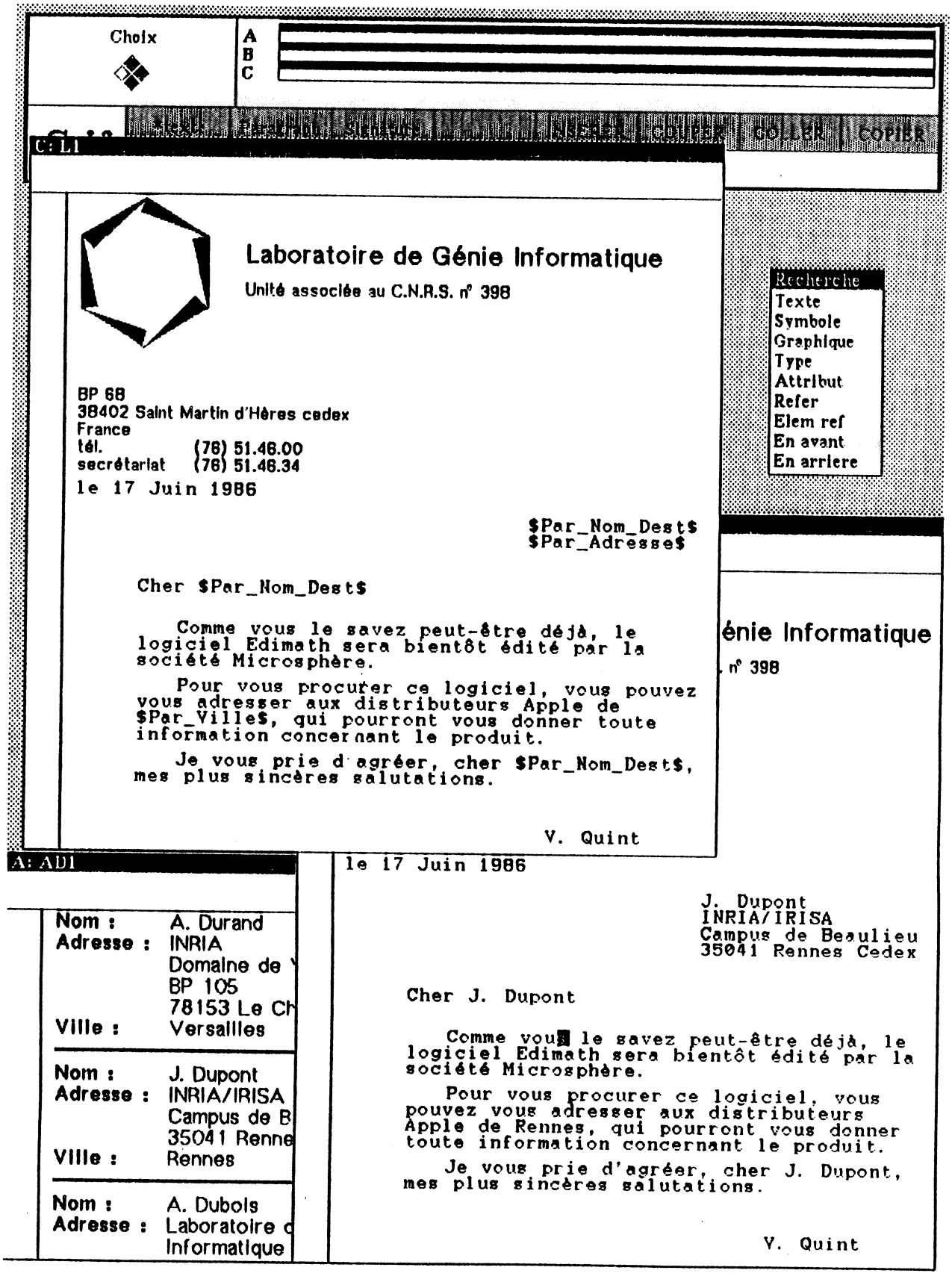


Figure 4.9 : L'image d'une lettre publi-postée.

Le menu affiché en haut à droite donne toutes les commandes de recherche. Elles permettent de rechercher une chaîne de caractères, un symbole mathématique, un élément graphique, un élément d'un type donné, un attribut une référence à l'élément sélectionné ou l'élément référencé par la partie sélectionnée. Les deux commandes suivantes permettent de poursuivre la recherche du même élément, en changeant éventuellement le sens de parcours du document.

La figure 4.10 montre un article dans une forme proche de la forme finale, celle qui est adoptée par la revue T.S.I. En plus de la vue principale du document, deux vues supplémentaires se trouvent sur l'écran : l'une montre la table des matières, l'autre les notes (on peut voir leurs références au début de l'introduction). On remarquera que les éléments qui figurent dans la table des matières sont présentés sous une forme différente de celle de la vue principale.

Le petit menu dans le coin inférieur droit de l'écran est celui qui apparaît lorsqu'on appelle la commande "Créer Vue". Ici, l'utilisateur a le choix entre les citations bibliographiques et les figures. On remarquera sur cette image un exemple d'imbrication d'objets de natures différentes (construits selon des structures génériques différentes) : le document (une structure générique) contient un tableau (une autre structure générique) qui contient à son tour des formules (troisième structure générique).

La figure 4.11 montre le même document que la figure précédente, mais avec un schéma de présentation différent, mieux adapté à la création du document, puisque certaines parties sont présentées sous la forme d'un formulaire, grâce à des boîtes de présentation. Notons que ce schéma n'affiche pas les éléments du type Photo_Auteur et qu'il place certains éléments dans un ordre différent par rapport au schéma précédent, notamment les domaines et les mots-clés. Ici une deuxième vue a été créée : elle donne toutes les formules numérotées du document. La vue des notes n'est pas créée, c'est pourquoi les deux références au début de l'introduction se présentent sous la forme d'une étoile entre crochets. Dès que la vue des notes sera créée, les références seront affichées sous la forme des numéros des notes auxquelles elles renvoient.

La troisième fenêtre affiche un autre document, une formule. C'est cette fenêtre qui a été utilisée pour la création de la formule de l'article. Une fois créée, la formule a ensuite été copiée et collée dans l'article, où elle peut encore être éditée, comme elle l'a été dans la première ligne du tableau.

Au moment où l'image a été prise, l'utilisateur mettait à jour la formule (1). Pour cela, avec la souris, il a sélectionné la racine (celle-ci s'est affichée en noir dans les deux vues où elle est visible) et il a appelé la commande "Insérer", avec l'option "Avant Racine" du menu d'insertion. Maintenant l'éditeur lui demande, sous la forme d'un menu, ce qu'il veut insérer avant la racine, puisque le schéma de structure des formules prévoit un

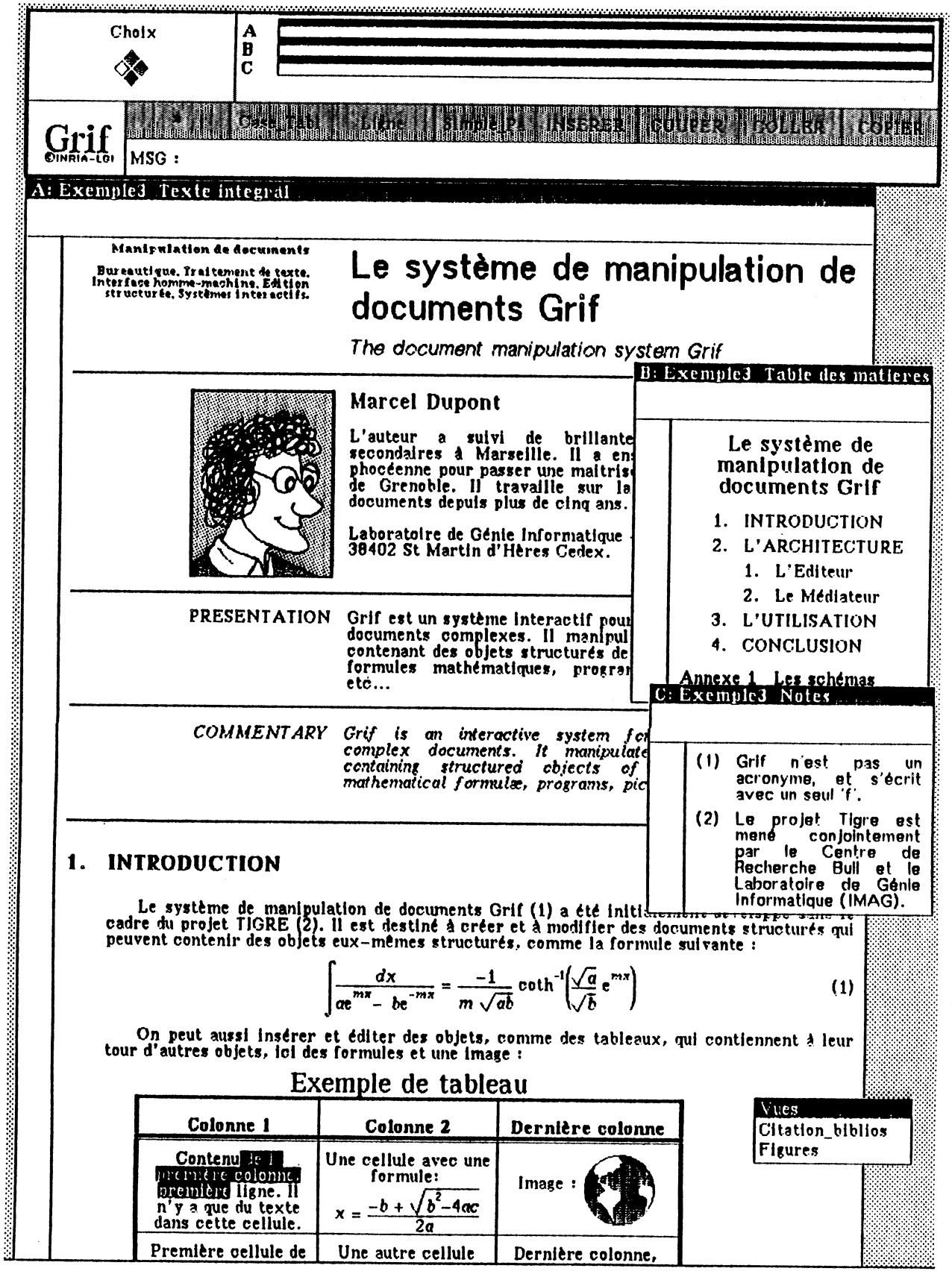


Figure 4.10 : L'image "finale" d'un article.

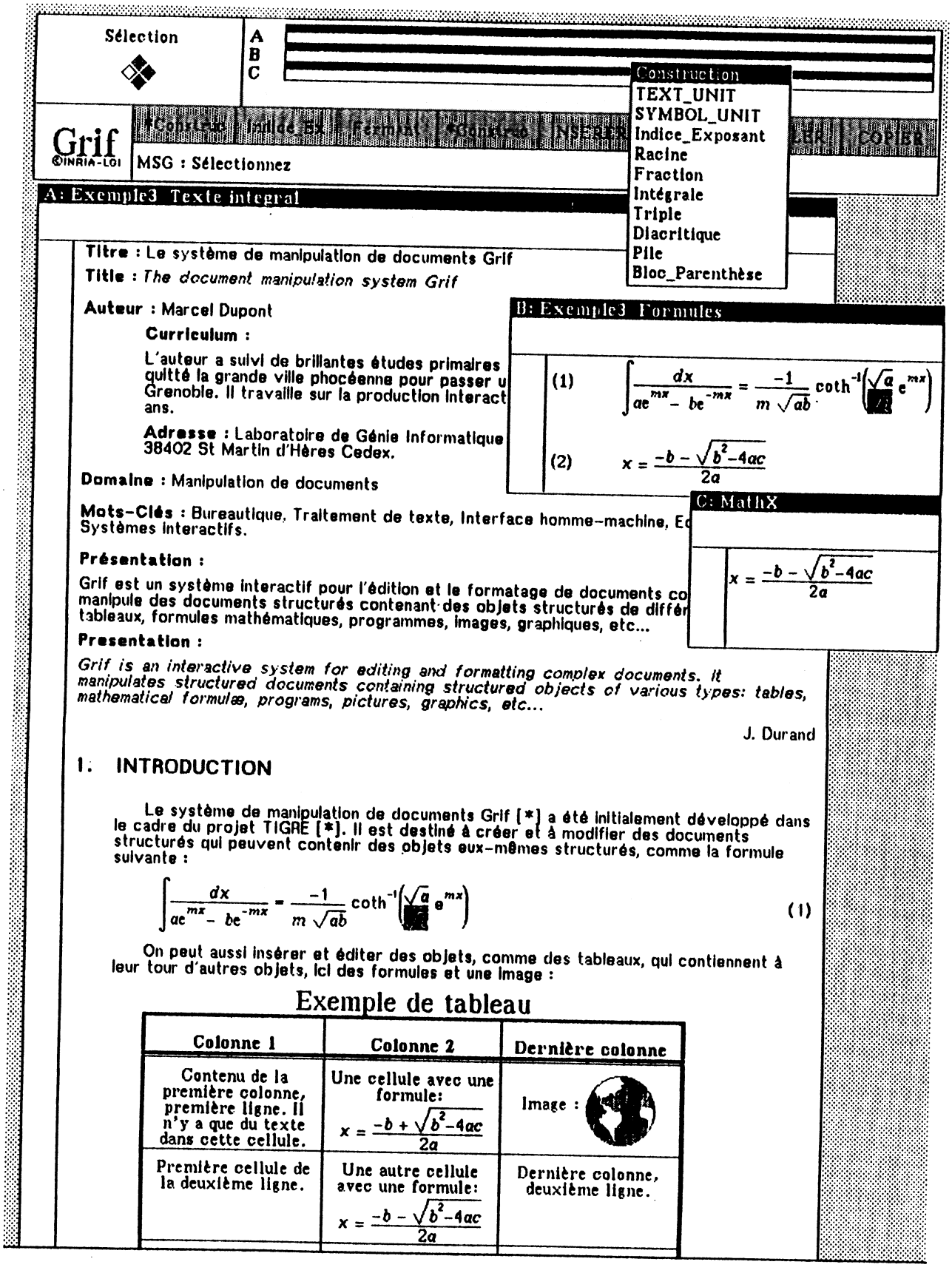


Figure 4.11 : L'image "de travail" d'un article.

choix entre plusieurs constructions. C'est ce choix qui est affiché dans le menu "Construction" sur le côté droit de l'écran. Ce type de menu est construit d'après le schéma de structure.

8.2. Une session d'édition

L'exemple développé dans cette section montre différentes étapes de la création et de l'édition d'un document de la classe "Article", dont le schéma de structure est donné à la fin du chapitre 3. Les documents des figures 4.10 et 4.11 ont également été construits selon ce schéma.

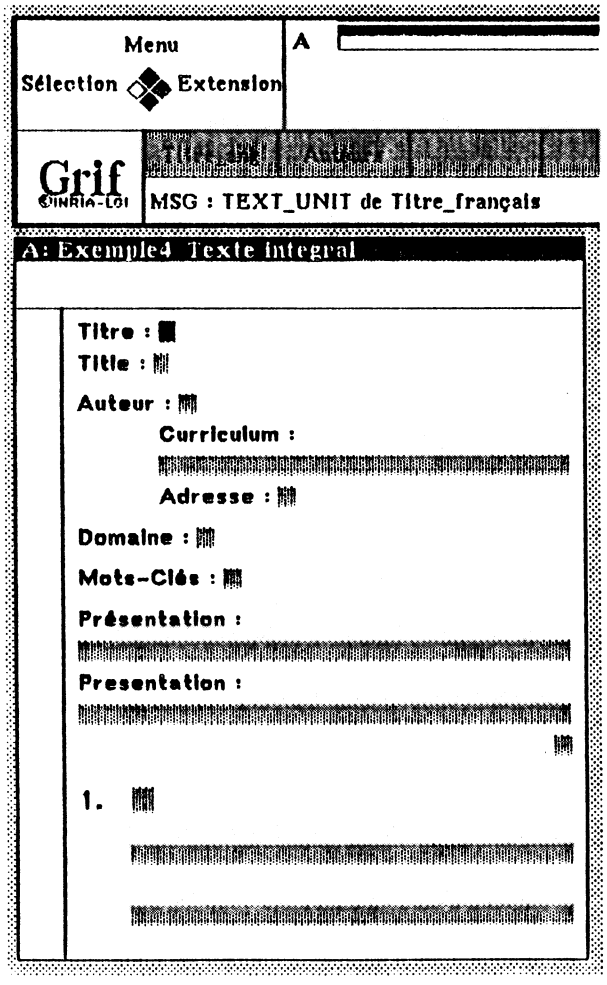


Figure 4.12.a

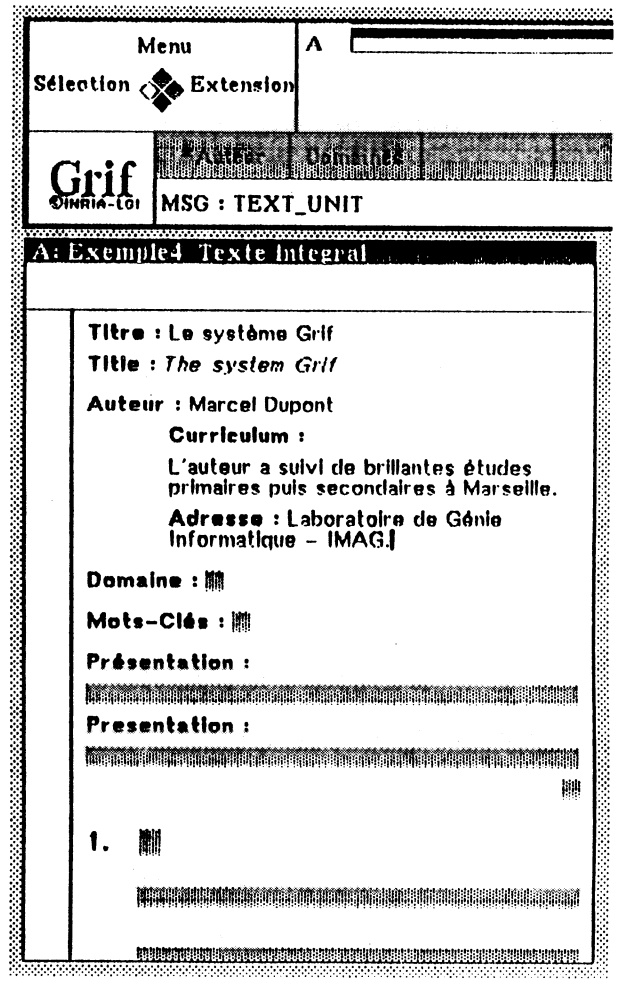


Figure 4.12.b

Lors de la création d'un nouveau document de cette classe, avec le schéma de présentation "de travail", l'éditeur affiche l'image de la figure 4.12.a, après avoir engendré les éléments spécifiés dans le schéma de structure. L'utilisateur remplit les premiers

éléments du document en sélectionnant à l'aide de la souris les rectangles gris qui représentent les éléments vides, puis en entrant leur texte au clavier. S'il les crée dans l'ordre indiqué sur l'image, il peut passer d'un élément au suivant en frappant simplement l'une des premières touches de fonction, dont l'image en haut de l'écran indique le nom de l'élément auquel on passe. Sur la figure, c'est le titre français qui est sélectionné. La première touche fait passer au titre anglais (élément suivant au même niveau dans l'arbre abstrait). La deuxième touche fait passer aux auteurs (élément suivant au niveau supérieur). Certains éléments peuvent évidemment être sautés et remplis plus tard.

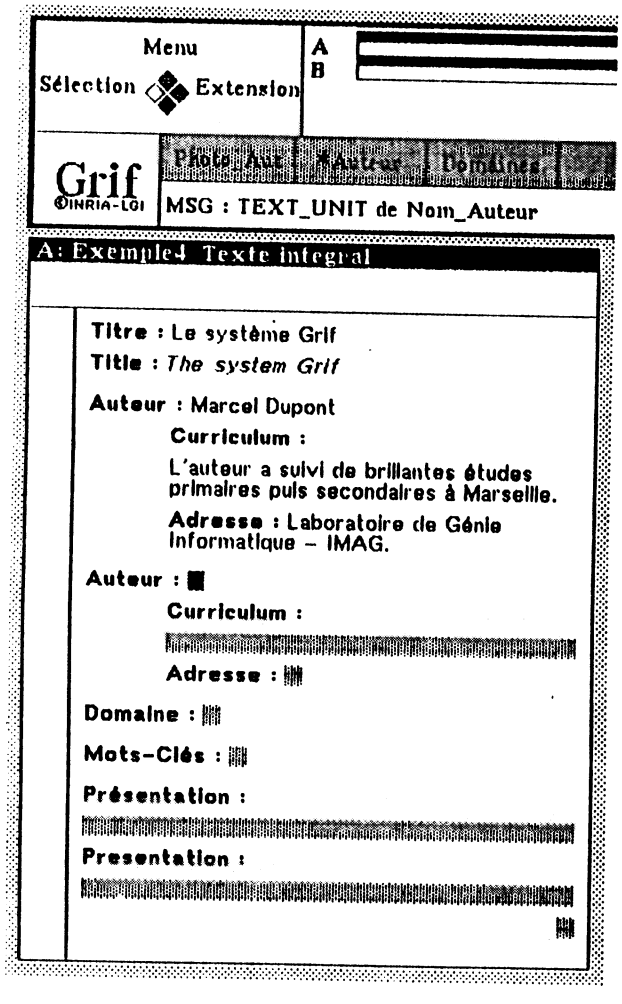


Figure 4.12.c

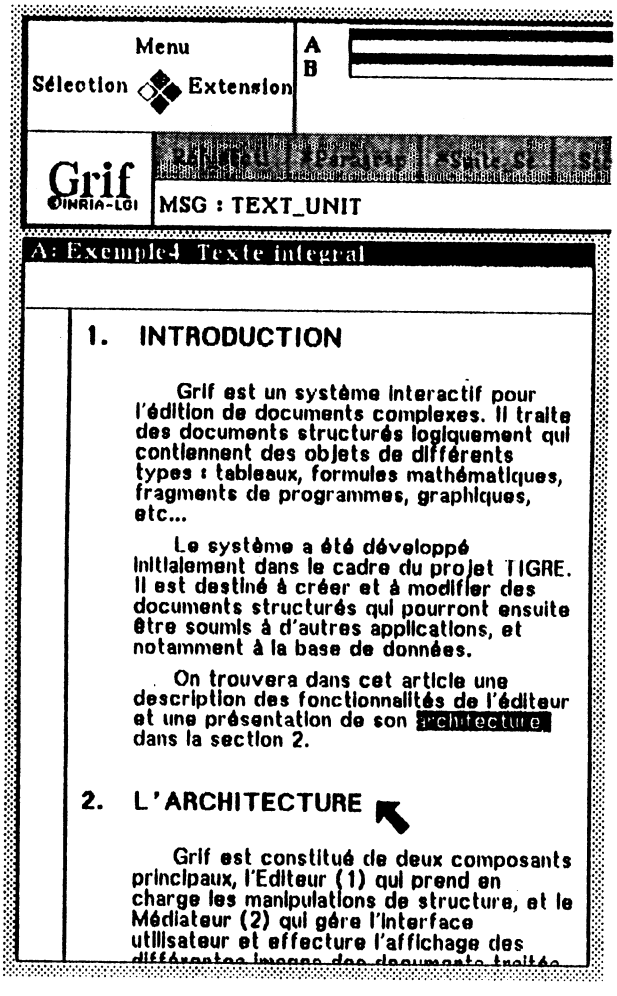


Figure 4.12.d

Après avoir rempli (partiellement) les champs concernant le premier auteur, (voir figure 4.12.b), il crée un deuxième auteur, soit en frappant la touche de fonction qui correspond à la case libellée "*Auteur", soit en appelant le menu d'insertion et en sélectionnant dans ce menu l'entrée "Après Auteur". Il obtient alors l'image de la figure 4.12.c. C'est la première feuille créée, le nom de l'auteur, qui est sélectionnée. Il reste à remplir, comme précédemment, les différents champs du nouvel auteur.

Sautons quelques étapes. La première section du document, l'introduction, se termine par une référence à la section 2, intitulée "L'architecture". Celle-ci commence par un paragraphe contenant deux références à des notes (de bas de page). Cela est présenté sur la figure 4.12.d (la vue des notes est affichée dans une autre partie de l'écran, non visible sur la figure). L'utilisateur veut alors ajouter une nouvelle section entre l'introduction et la section 2. Pour cela, il sélectionne en cliquant, par exemple, derrière le titre de la section 2, là où est la flèche sur la figure. Le titre apparaît alors en inversion video. Il entre la commande "Insérer" et choisit l'option "Avant Section" dans le menu qui lui est proposé. L'éditeur affiche l'image de la figure 4.12.e. On peut noter sur cette image que la section sur l'architecture a changé automatiquement de numéro et que la référence à cette section (à la fin de l'introduction) a été mise à jour en conséquence.

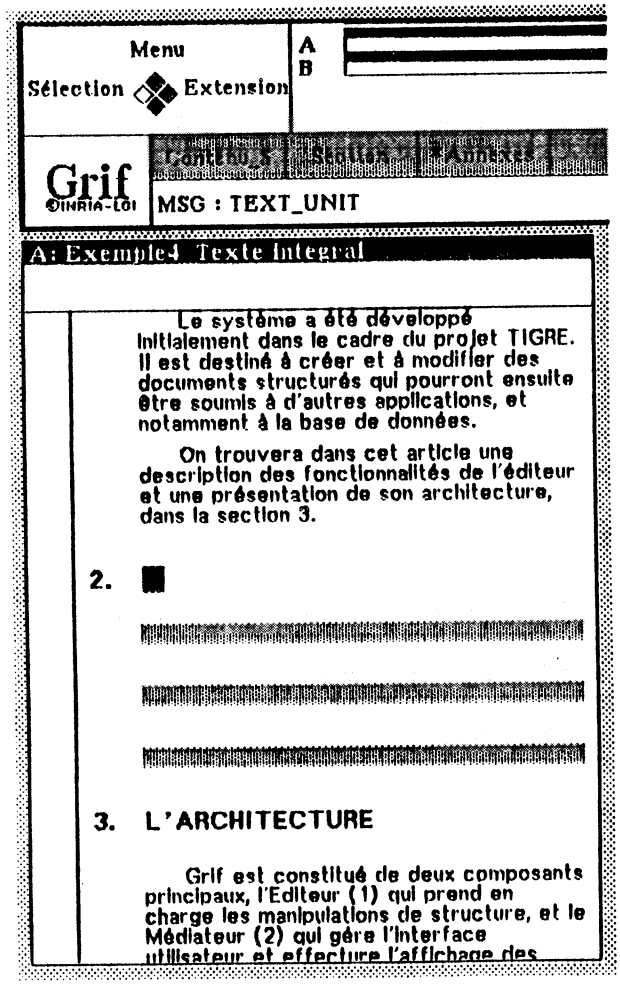


Figure 4.12.e

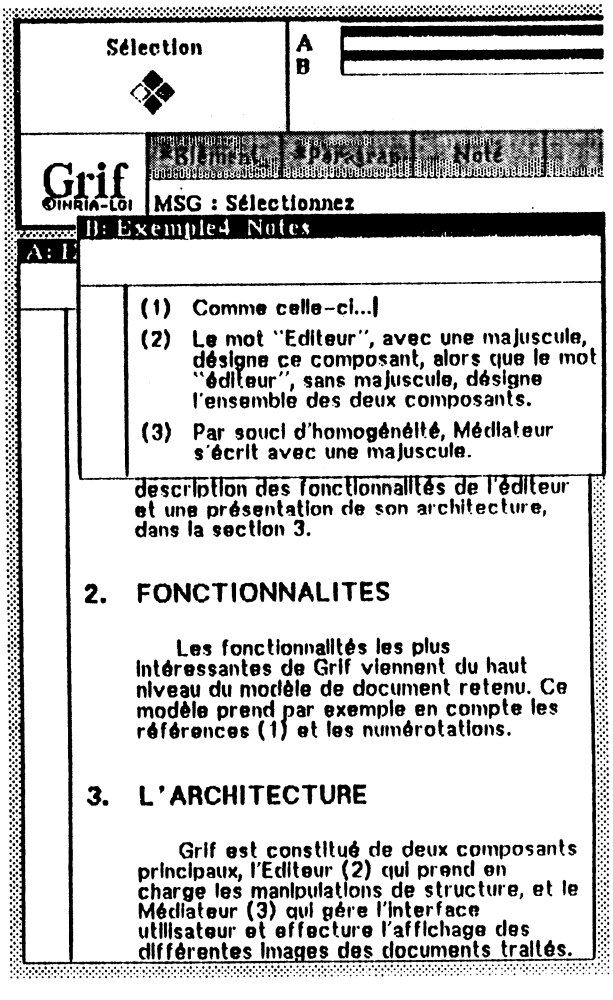


Figure 4.12.f

Les trois rectangles gris qui apparaissent sous le titre de la nouvelle section représentent son contenu minimum : le premier paragraphe de la section, suivi de deux sections de niveau inférieur, qui sont vides : l'éditeur n'a pas créé leur contenu puisque leur

définition est récursive. L'utilisateur est libre de détruire ces deux sections ou au contraire de demander à l'éditeur d'en créer le contenu.

Sur la figure 4.12.f, l'utilisateur a effectivement détruit les deux sous-sections et il a tapé le titre de la nouvelle section 2 et le texte du premier paragraphe. Dans ce paragraphe, il a introduit une note. Pour cela, il a déplacé la fenêtre de la vue des notes, il a sélectionné la première note dans cette vue, puis il a appelé la commande "Insérer" avec l'option "Avant Note". Une nouvelle note est apparue en haut de la vue des notes, avec le numéro 1, et les notes qui existaient déjà ont été automatiquement renumérotées, leurs références étant mises à jour simultannément. Après avoir entré le texte de la note, l'utilisateur a créé une référence (commande "Insérer") et désigné la nouvelle note avec la souris. Le numéro de la note est alors apparu dans le texte pour représenter la référence.

9. DISCUSSION

9.1. L'intégration

L'un des objectifs principaux du projet Grif était de construire un système *intégré*, c'est à dire un système où l'utilisateur évolue dans un environnement unique, quelle que soit la nature des informations qu'il manipule : structure de document, texte, formule, table, etc... Cet objectif a effectivement été atteint, mais avec un plus ou moins grand succès selon les natures d'information.

L'intégration des manipulations de structure et de texte, ou plus généralement de contenu, a été obtenue grâce à l'architecture choisie. Le Médiateur réalise les fonctions habituelles d'un traitement de texte et appelle lui-même l'Editeur pour le traitement des structures. Les deux composants se répartissent le travail sans que l'utilisateur ait à indiquer explicitement la tâche de chacun. Un autre facteur facilitant cette intégration est le choix des commandes, qui s'appliquent aussi bien au contenu qu'à la structure, accentuant ainsi l'homogénéité de l'interface.

A la différence d'Edimath, Grif offre une intégration complète des formules mathématiques dans les documents. En effet, les principaux défauts d'Edimath viennent du changement d'environnement pour l'édition du texte et de la difficulté d'éditer une formule qui a déjà été insérée dans un document. Ces défauts n'apparaissent pas dans Grif, où il est toujours possible de passer d'une façon harmonieuse d'une formule au reste du document, et inversement. De plus les formules se marient bien au texte : il n'y a aucune difficulté à mettre des expressions mathématiques dans le fil du texte. Ce type d'intégration a été obtenu par la généralité des modèles, de structure comme de présentation, qui

permettent de décrire et de traiter des objets de différentes natures avec des moyens identiques.

Pour les mêmes raisons, le même type d'intégration est obtenu pour les tables, qui peuvent s'insérer dans les documents, mais qui peuvent aussi contenir d'autres objets. Cependant, une difficulté se pose avec les tableaux complexes, dont la structure générique est difficile à décrire à l'aide des constructeurs existants. Mais pour les tables relativement simples, l'intégration est totale : on peut couper un fragment de texte dans un paragraphe et le coller dans une cellule d'un tableau, on peut procéder de la même façon avec une expression mathématique (voir figure 4.11).

Dans le domaine du graphique Grif se montre moins efficace. Sauf pour quelques types de graphiques très structurés, il est souvent plus facile d'utiliser des outils comme MacPaint ou MacDraw [Williams]. Mais Grif offre toujours la possibilité d'insérer dans un document, même à l'intérieur d'un texte mis en lignes, un graphique ou une image créée par un autre moyen. La seule difficulté est alors de modifier cet élément dont la structure n'est pas connue de Grif. On est ramené au même type d'intégration que sur le Macintosh.

9.2. L'interface utilisateur

Un autre objectif du projet était de concevoir et de réaliser une interface utilisateur qui permette de travailler directement sur l'écran, en évitant les références explicites aux arborescences. Les nombreux paramètres de présentation de Grif permettent d'utiliser largement les effets graphiques pour montrer la structure logique : les changements de corps ou de style des caractères, les justifications, les positions des éléments sont autant d'indications sur la structure.

Grâce à la richesse graphique des images, l'utilisateur se repère vite dans la structure. De plus les moyens de sélection par désignation directe à l'écran permettent, avec un petit nombre de commandes, de créer de nouveaux éléments structurels ou de parcourir le document par sa structure. On obtient ainsi la simplicité d'utilisation des systèmes WYSIWYG sans pour autant perdre les avantages d'une représentation de haut niveau.

Cependant l'analogie avec les systèmes WYSIWYG s'arrête là. L'image présentée sur l'écran ne montre en effet pas de pages et tous les éléments n'occupent pas la position qu'ils auront à l'impression ; c'est notamment le cas des éléments associés comme les notes ou les figures. Cela ne pose d'ailleurs pas de difficulté, puisque ces composants sont traités lors de l'impression comme des éléments flottants, c'est à dire sans position prédéterminée dans le texte. Il est même souvent préférable, pendant l'édition, de pouvoir les placer librement sur l'écran, simplement en déplaçant leur fenêtre, pour les afficher plus près de la partie de texte qui y fait référence.

Une conséquence intéressante de l'absence de pages est que les temps de calcul pour le réaffichage des parties modifiées est indépendant du volume du document, ce qui garantit des temps de réponse constants tout au long de l'édition d'un gros document.

9.3. La structure

Une des difficultés du projet était de trouver les bonnes structures à mettre en œuvre, sans enfermer l'utilisateur dans un carcan trop rigide et tout en structurant autant que nécessaire les documents qui doivent l'être.

Ce compromis entre la rigueur des structures et une certaine souplesse a été trouvé avec les constructeurs des structures génériques, mais aussi dans le comportement de l'éditeur qui propose plus qu'il n'impose. Tous les éléments prévus dans un schéma de structure sont engendrés automatiquement mais l'utilisateur est libre de les laisser vides ou même de les supprimer. L'éditeur est aussi capable de recréer les éléments qui auraient été détruits par erreur.

Le principe même des structures génériques, qu'un utilisateur peut créer ou modifier, participe aussi à la souplesse du système. On résoud ainsi l'un des problèmes posés par Edimath, où les constructions mathématiques absentes du modèle ne peuvent être ajoutées simplement.

Toutes les structures manipulées par Grif sont des arbres. On y a ajouté des liens non hiérarchiques pour résoudre le problème des références, mais sans boulverser les structures de données essentiellement arborescentes. Ces structures ont prouvé leur efficacité dans la représentation des documents et des objets qu'ils contiennent. Toutefois, comme on l'a vu plus haut, d'autres structures pourrait être utiles pour représenter des tableaux complexes et certains types de graphiques.

9.4. Les lacunes

Bien que Grif comporte l'essentiel des caractéristiques nécessaires pour manipuler efficacement des documents structurés, quelques développements complémentaires seraient encore utiles.

Actuellement le système ne permet pas réellement de traiter un gros document sous la forme de plusieurs documents de taille plus réduite. C'est pourtant, en général, la façon dont sont écrits des documents tels que les rapports divisés en chapitres. En fait, rien n'empêche, dans l'état actuel de Grif, de réunir plusieurs documents contenant chacun un chapitre pour constituer un rapport, à condition qu'il n'y ait pas de référence d'un chapitre

vers d'autres. Les références entre documents ne sont pas prévues, et c'est cela qui interdit de fait le traitement par parties des documents volumineux.

On a vu que les documents produits par Grif pouvaient être traduits automatiquement de façon à être soumis à un formateur ou transmis selon un standard de représentation de documents. Mais si Grif est largement ouvert en sortie, les possibilités d'entrée de documents produits par d'autres systèmes sont encore limitées. Le problème peut se résoudre de deux façons différentes selon le type de représentation du document à entrer. S'il s'agit d'une représentation de haut niveau, comparable à celle de Grif, le problème se ramène grossièrement à un changement de syntaxe et un outil comme le traducteur de Grif, mais fonctionnant en sens inverse, peut être développé. On peut envisager d'accepter de cette manière des documents venant de Scribe, LaTeX, SGML, ODA (niveau logique). Si au contraire il s'agit d'une représentation physique (PostScript, T.73 ou même simple texte ASCII), il faut reconnaître la structure à partir de la présentation ou du contenu du document. Certains travaux vont dans ce sens [Tazi, Verges], mais il ne semble pas que le problème soit encore totalement résolu.

Bien que l'accent ait été mis sur l'interface utilisateur, Grif ne dispose pas d'un système général d'aide, comme on en trouve dans d'autres éditeurs, comme Etude notamment [Hammer]. Il est probable qu'un tel outil pourrait permettre à un utilisateur novice d'acquérir plus rapidement une bonne maîtrise du système. Pour notre part nous avons préféré concentrer nos efforts sur l'affichage et les images abstraites, qui participent largement au confort de l'utilisateur [Quint 87b].

L'utilisation de Grif nous a montré que la conception des schémas de présentation était une tâche délicate. Un langage compilé ralentit le cycle de mise au point des schémas, alors que de nombreux essais sont nécessaires pour trouver les bonnes relations entre les boîtes des éléments d'une classe de document. Un système interactif pour la création de ces schémas serait un avantage certain. Dans un tel système, le principe serait, comme dans Juno [Nelson], d'exprimer graphiquement, sur l'écran, des contraintes entre éléments (boîtes) tout en montrant immédiatement les conséquences sur les autres éléments. Cet outil engendrerait directement les tables de présentation utilisées par l'éditeur Grif.

Enfin, la connexion à un système de bases de données adapté [Velez] n'a pas encore pu se réaliser. Il est clair qu'un outil comme Grif tirerait un avantage certain d'un système de gestion de bases de données spécifiquement adapté aux documents structurés. Un tel système pourrait prendre en charge la gestion des versions, les relations entre documents, il pourrait gérer les schémas de structure et de présentation. Il pourrait jouer le rôle d'un système documentaire pour les documents produits par Grif. C'est dans cette perspective que nous n'avons introduit aucune organisation dans l'ensemble des fichiers utilisés par Grif (représentation pivot des documents, schémas sources et compilés).

CHAPITRE 5

CONCLUSION

En guise de conclusion, on présente dans ce chapitre les différentes utilisations qui peuvent être faites d'un système tel que Grif. Grif est ensuite comparé à un certain nombre de systèmes analogues et on montre en quoi il s'en distingue. Enfin, à partir de l'expérience tirée de Grif et de l'étude des autres systèmes, on propose un scénario pour l'évolution des systèmes d'édition de documents structurés.

1. LES DOMAINES D'UTILISATION

Grif peut être utilisé dans différents types d'applications. Cette section indique quelques utilisations possibles de l'éditeur, tout en montrant l'intérêt d'une approche à la fois structurée et interactive.

1.1. Un générateur d'éditeurs

On a vu avec Edimath que la spécification de la structure logique des objets que doit manipuler un éditeur était une tâche délicate. Il en est de même pour la mise au point des règles de présentation, où l'on procède par retouches successives. Dans Edimath, le modèle de structure logique et les règles de présentation sont figés dans le programme lui-même, ce qui permet d'obtenir de bonnes performances sur des machines modestes, mais rend particulièrement lourd tout changement des règles et du modèle. Un outil comme Grif aurait été très utile pour mettre au point une maquette, qui aurait pu ensuite orienter la réalisation de l'éditeur final pour Macintosh.

Si Grif ne peut pas être utilisé sur de petites machines (il lui faut une taille mémoire conséquente et une bonne puissance de calcul), il peut servir à construire très rapidement des maquettes d'éditeur structuré pour ces machines. En tirant partie de la souplesse des schémas de structure et de présentation, il est facile d'essayer plusieurs types de structures et de varier les présentations jusqu'à obtenir celle qui convient le mieux à une utilisation particulière. De plus, la variété des commandes disponibles dans Grif permet d'évaluer, pour une structure de document donnée, quel est le jeu minimum de commandes à offrir à l'utilisateur. On peut ainsi réaliser rapidement une maquette évolutive qui permet de spécifier précisément un éditeur structuré pour une petite machine.

Grif peut servir à traiter des documents très divers, comme on l'a vu tout au long des chapitres précédents, mais, pour une catégorie d'utilisateurs déterminée, il peut être adapté à une seule application, avec un seul type de document ou d'objet. En spécifiant un seul schéma de structure et un ou plusieurs schémas de présentation associés, il se présentera à ses utilisateurs comme un éditeur spécialisé dans le traitement de leurs documents habituels. De cette façon, chaque groupe d'utilisateurs peut avoir son éditeur spécialisé, tout en conservant de bonnes possibilités d'évolution de l'outil.

1.2. Une aide à la saisie

Grif peut être considéré comme un système autonome qui utilise un ou plusieurs formateurs pour imprimer ses documents, grâce au traducteur qui l'accompagne. C'est le point de vue qui a été adopté dans les chapitres précédents, où l'éditeur est présenté comme le composant essentiel. Mais on peut aussi considérer les choses à partir du formateur. Une organisation peut très bien faire le choix d'un formateur, en raison de ses qualités et avantages propres, et considérer Grif comme un outil permettant de saisir et de modifier des documents destinés à ce formateur.

Si le formateur n'est considéré que comme un outil de sortie banal de Grif et que son choix n'est pas fait *a priori*, on conçoit la structure générique des documents et leur présentation en prenant d'abord en compte le type d'information que le document devra véhiculer. Les contraintes du formateur n'interviennent pas dans cette phase de conception des documents, et c'est seulement après que l'on cherche à traduire l'organisation des documents dans les termes d'un ou plusieurs formateurs.

A l'inverse, si les documents traités par Grif sont destinés d'abord à un formateur particulier dont on veut utiliser toutes les ressources, on conçoit les structures des documents de sorte qu'elles soient très proches ou même identiques à celles acceptées par le formateur. On peut aussi définir une présentation qui ressemble au mieux à ce que produira le formateur. Ainsi, l'utilisateur du formateur peut enfin voir ce qu'il fait, détecter (et même éviter) la plupart des erreurs qu'il commettait précédemment, et modifier son document sur une forme lisible.

Il est clair que cette façon de faire est particulièrement intéressante avec des formateurs qui considèrent les documents à un niveau d'abstraction assez élevé, puisqu'ils ont alors des modèles de document voisins de celui de Grif. Les expériences faites avec Mint [André 86] et LaTeX [Quint 86b] l'ont bien montré. Au contraire, des formateurs de plus bas niveau, comme Troff ou TeX sont plutôt mieux adaptés à la première approche ; ils permettent de traduire simplement la structure des documents en termes de commandes

typographiques ou de mise en page, un peu comme lorsqu'on écrit des macro-commandes dans ces formateurs.

1.3. La production de documents normalisés

Les structures logiques des documents de Grif et le comportement de l'éditeur lui-même peuvent apporter une aide efficace dans la création de documents normalisés. Des documents structurés d'une façon très rigide, comme des formulaires ou des bordereaux (voir figure 4.8) ou encore les en-têtes de certains documents (voir figure 4.11) peuvent être saisis sans omettre aucun champ. De plus, les documents produits par Grif peuvent être exploités par d'autres applications qui sauront, grâce à la structure, retrouver dans le document les informations qui les intéressent. Ainsi ces documents pourront être non seulement créés et imprimés, mais aussi classés, transmis, recherchés, indexés, etc...

La normalisation peut concerner l'organisation logique des documents, mais aussi leur représentation externe. Les standards de représentation de documents prennent de l'importance [Joloboff 86, Joloboff 87], et les nouveaux systèmes de traitement de documents devraient à terme s'y conformer. Grif ayant été conçu et réalisé pendant l'élaboration de ces normes, les documents qu'il produit ne les respectent évidemment pas, mais les concepts de son modèle de document sont voisins de ceux de ODA [Horak] et plus encore de SGML [Goldfarb]. Il peut être adapté à ces standards. Une expérience concluante a déjà été réalisée, avec l'utilisation du traducteur, pour produire des documents conformes à l'utilisation de SGML définie par l'AAP [AAP].

1.4. Les métiers du livre

Si Grif peut être utilisé dans un contexte bureautique, où une même personne, avec un seul outil, conçoit, écrit, modifie, présente, met en page, et imprime ses documents, il peut aussi rendre des services appréciables dans la chaîne de production traditionelle des ouvrages imprimés, où, au contraire, plusieurs professionels interviennent avec des outils différents dans le cycle de production des documents.

Dans cette chaîne complexe, l'éditeur (dans le sens du mot anglais *publisher*) détermine les structures génériques ; le maquettiste définit la présentation et le typographe écrit les règles de traduction dans le langage d'entrée de son système de composition. L'auteur utilise Grif avec les schémas élaborés par les précédents. Ainsi chacun peut jouer pleinement son rôle sans être perturbé par les autres intervenants, et tous peuvent travailler sur la même représentation du document.

L'auteur peut se concentrer sur le contenu et l'organisation de son document. Il ne se préoccupe pas de la présentation, et n'a donc pas à remplir les fonctions du maquettiste et du typographe, dont il ignore le plus souvent les règles de l'art. L'éditeur est sûr de recevoir un document conforme aux normes de son journal ou de sa collection et il peut communiquer avec l'auteur en annotant le document, sous sa forme électronique. Le typographe n'est pas dérangé par les caractères gothiques que l'auteur trouve si beaux ou par les changements continuels des marges, les retraits et centrages variés, les numérotations bizarres et autres détails "typographiques". Il trouve toute l'information dont il a besoin (la structure logique, les intentions d'auteur) et n'a plus à débarasser le document de toutes les scories devenues habituelles avec le Macintosh.

Dans une telle organisation du travail, l'éditeur Grif ne remplit qu'une partie des fonctions. Mais les documents qu'il produit peuvent être repris par d'autres outils, comme les programmes de mise en page, de photocomposition ou d'indexation. Il est certain que la mise en conformité de Grif avec une norme comme SGML en augmenterait encore l'intérêt dans la chaîne éditoriale. L'expérience évoquée plus haut est un pas dans ce sens.

1.5. La documentation technique

La documentation technique pose de nombreux problèmes, qui ne sont pas tous résolus par les systèmes actuellement mis en œuvre dans ce domaine. En effet, des outils comme Interleaf, qui se veulent orientés vers la documentation technique, permettent principalement l'intégration de différents types de graphiques dans les documents, avec un contrôle aisé à l'écran. Mais les documents techniques se caractérisent aussi par une structure très forte qui est utilisée pour les numérotations, les renvois, les tables et index, etc... Dans ce domaine Grif offre les principales fonctionnalités nécessaires.

Le classement et la consultation des documents techniques présente également des difficultés. Ces documents ne sont généralement pas faits pour être lus séquentiellement, mais bien pour être accédés par des moyens comme les index, les tables des matières ou les renvois. Grif peut servir à parcourrir efficacement ce genre de documents. La richesse de sa structure logique autorise à stocker les documents qu'il produit dans des bases de données adaptées, où ils pourront ensuite être retrouvés aisément. Dans des secteurs d'activité comme l'aéronautique ou le nucléaire, la recherche du document pertinent parmi une masse énorme de papier est un vrai problème. Grif propose un élément de solution.

2. COMPARAISON DE GRIF AVEC D'AUTRES SYSTEMES

Nous comparons ici Grif avec quelques systèmes décrits dans la littérature et qui, à un titre ou à un autre, ont suscité un certain intérêt. N'ont été retenus pour cette comparaison que les systèmes interactifs qui prennent en compte l'aspect logique des documents, même s'il ne s'agit pas toujours d'une structure très forte. On ne trouvera donc ici ni formateur ni système de traitement de texte purement WYSIWYG.

Parmi les systèmes retenus, trois sont relativement anciens, puisqu'ils ont été présentés en 1981 ; ce sont Etude [Hammer], PEN [Allen] et [Walker]. La machine Lilith a donné donné lieu à deux éditeurs intéressants : Andra [Gutknecht 84] et Lara [Gutknecht 85]. Deux systèmes développés plus récemment en France ont également été considérés : Mentor-Rapport [Mélèse] et Pléiade [Nanard 86]. Enfin, on a retenu tous les systèmes de la classe considérée qui ont été présentés en même temps que Grif [Quint 86a] à la conférence EP'86 : [Hamlet], [Cowan], [Coray], [King 86] et [Furuta 86a]. Il faut d'ailleurs souligner que l'édition interactive de documents structurés constituait le thème dominant de la conférence.

Après une très brève présentation de chacun de ces systèmes, nous les examinerons globalement, en fonction d'un certain nombre de caractéristiques communes.

2.1. Les systèmes retenus

PEN [Allen] est un système pour la préparation de manuscrits contenant des notations mathématiques. Il est orienté vers la saisie d'un document structuré et se repose sur un formateur pour l'impression des documents qu'il produit. Le traitement des formules se fait à partir d'une notation très concise, basée sur APL, qui permet à la fois le formatage et l'évaluation des formules.

Etude [Hammer] se présente comme un système interactif d'édition et de formatage de documents et met l'accent sur l'interface utilisateur et la facilité d'apprentissage et d'utilisation de l'outil. Il travaille sur un écran à matrice de points et cherche à afficher une image des documents de qualité typographique. Cette image est construite d'après la structure : le document est typé et est formé de composants également typés. Comme Scribe, Etude utilise une base de données qui définit les formats des différents types d'éléments.

Le système décrit par [Walker] est très influencé par les environnements de programmation ; il est principalement destiné aux documents techniques. En fait deux systèmes sont présentés : l'un est l'outil idéal (selon l'auteur) dont devraient disposer les rédacteurs techniques, l'autre est celui qui a été réalisé ; c'est un sous-ensemble du premier.

Les documents ont une structure très riche et le système peut effectuer des traitements puissants (recherche des références, constitution d'index, etc...). Malgré cela, la structure peut être rendue transparente à l'utilisateur. En plus de l'édition structurée proprement dite, le système propose des outils d'aide à l'écriture, comparables à ceux présentés par [Cherry], et des outils de gestion des documents : index, relations entre documents, historiques, commentaires...

Andra [Gutknecht 84] est le système de préparation de documents qui a été développé spécifiquement pour la machine Lilith. Sa conception a été nettement influencée par Bravo [Lampson], dont il reprend l'interface utilisateur : fenêtres, menus, souris. Andra affiche en permanence l'image finale du document, bien que les pages ne soient pas visibles à l'écran. Un document est structuré de façon arborescente, mais l'utilisateur n'a aucun contrôle sur cet arbre, et ne le perçoit pas. A chaque document est associé un fichier de style qui contient les attributs de présentation des différents types d'éléments constituant le document.

Lara [Gutknecht 85] est le successeur d'Andra. Il a été développé également sur Lilith et remédie à certains défauts d'Andra. Il s'en distingue essentiellement sur deux points : la structure logique et la présentation. La structure logique des documents est limitée à quatre niveaux "universels" : document, chapitre, paragraphe, chaîne de caractères. Chaque niveau a un ensemble spécifique d'attributs de présentation, par exemple à un chapitre sont associés des marges de haut et de bas de page ou un en-tête de page ; à un paragraphe des marges droite et gauche, un mode de justification ou un retrait de la première ligne ; à une chaîne de caractères, une police ou un espacement. Autre différence par rapport à Andra, ces attributs de présentation se trouvent dans le fichier du document et sont donc spécifiques à un document.

Mentor-Rapport [Mélèse] est très proche des environnements de programmation, puisqu'il est constuit sur Mentor [Donzeau]. Son domaine d'utilisation est d'ailleurs plutôt orienté vers le génie logiciel. Il hérite des qualités de Mentor : il est programmable, extensible, ouvert et accepte plusieurs formalismes. Il en subit également les lourdeurs, comme le changement d'environnement très net entre les manipulations de texte et de structure. L'extension réalisée par rapport à Mentor atténue toutefois ces effets, puisque des commandes spécifiques ont été ajoutées pour faciliter le traitement des documents et rendre l'interface utilisateur accessible à des non-programmeurs. Mentor-Rapport propose un modèle unique et général à partir duquel tout document doit être construit. Il fournit les types chapitre, section, sous-section, paragraphe, annexe, bibliographie, etc...

Pléiade [Nanard 86] est très nettement inspiré du système d'Interleaf [Morris 85] dont il reprend le modèle de document relativement peu structuré. Il offre des fonctionalités semblables sur des machines plus répandues. C'est un système facile à utiliser dont l'interface est à base d'icônes et de menus arborescents. Il présente une image identique à

ce qui sera imprimé, et prend lui-même en charge l'impression des documents. Il contient un éditeur spécialisé pour les formules mathématiques, très proche d'Edimath. On peut le définir comme un système WYSIWYG dont la présentation est basée sur le typage des éléments constituant le document.

Le système de [Hamlet] n'est pas réalisé. Il s'agit plutôt de spécifications. Il propose d'utiliser les techniques du génie logiciel, notamment la conception descendante et les environnements intégrés, pour améliorer la production des documents. L'accent est mis sur la structure qui est considérée à un niveau assez fin. La présentation des documents est décrite par des règles qui sont appliquées par le système, mais l'utilisateur peut encore intervenir pour modifier la mise en page finale. Le système engendre un fichier de commandes typographiques, l'impression étant effectuée à l'extérieur. L'influence des environnements de programmation se fait sentir dans l'interface utilisateur qui risque de rebuter le non-programmeur.

De même que le précédent, le système proposé par [Cowan] n'est pas encore réalisé. Ses objectifs sont très proches de ceux de Grif et de nombreuses caractéristiques sont communes. En particulier, le modèle de document est très voisin et repose sur la même notion de classe. La saisie du document est guidée par la structure de la classe. L'image est construite à partir de la structure et plusieurs présentations sont possibles pour une même classe ; l'image affichée sur l'écran est seulement proche de l'image qui sera imprimée. L'interface utilisateur rend l'apprentissage facile et rapide. L'impression se fait par l'intermédiaire d'un formateur.

L'objectif de [Coray] est de combiner les avantages des systèmes interactifs et des formateurs. Le système travaille à partir d'une spécification de la structure du document, qui sert de guide pour l'édition et le formatage. La spécification descriptive du formatage est séparée du contenu du document et est effectuée par un utilisateur spécialisé. Des éléments comme des formules, des figures, des fragments de programmes peuvent être intégrés aux documents. Le système présente simultannément une image formatée, donnant l'aspect final du document, et une image plus grossière (une seule police, chasse fixe) sur laquelle l'utilisateur agit. Un environnement multi-tâche gère le processus d'édition et plusieurs processus de formatage qui se déroulent en parallèle.

Le système W [King 86] est un développement du prototype W-p [King 84]. Il est également basé sur la notion de classe de documents définie par une grammaire. Le système essaie de se conformer le mieux possible à la philosophie WYSIWYG. Le reformatage n'est pas automatique et l'utilisateur décide de la partie du document à réafficher après les modifications. L'impression est prise en charge par le système qui ne gère actuellement qu'une imprimante (Postscript et TEX sont prévus dans de prochaines versions). Des tables, des images et des formules peuvent être intégrées au document. Les formules mathématiques sont traitées de la même façon que n'importe quel élément du

document. Enfin W a déjà été porté sur différentes machines, qui vont du Macintosh au VAX, en passant par une station de travail sous UNIX.

Le système développé par R. Furuta est présenté succintement dans [Furuta 86a] et d'une façon plus détaillée dans [Furuta 86b]. Il s'agit du prototype d'un système représentant les documents comme des objets abstraits tout en fournissant un mode de manipulation "naturel", c'est à dire interactif. La structure interne des documents est hybride : il y a au plus haut niveau un arbre, dont les feuilles peuvent elles-mêmes être structurées, mais avec des structures spécifiques, pas nécéssairement arborescentes, et variables selon le type de feuille (texte, tableau, formule mathématique). Des éditeurs spécialisés prennent en charge la manipulation du contenu de ces feuilles, alors qu'un éditeur généralisé traite les arbres. Il semble que l'utilisation de terminaux alpha-numériques ait orienté le mode de présentation des documents, que l'utilisateur voit sous la forme de gabarits représentant tous les éléments qu'il peut créer ou qui existent déjà. L'image est donc assez différente de ce qui sera imprimé. L'impression est prise en charge par un formateur indépendant de l'éditeur.

2.2. La structure logique

Tous ces systèmes prennent en compte la structure logique des documents qu'ils manipulent, mais pas tous de la même façon, ni dans le même but, ni au même niveau.

Seuls deux systèmes, Lara et Mentor-Rapport, imposent un modèle unique et général, que tout document est censé suivre. Ce modèle définit les types des éléments qui doivent constituer un document et n'est pas extensible par l'utilisateur. Il est probablement mal adapté et trop contraignant dès qu'on veut traiter autre chose que des documents techniques.

Avec Pléiade, au contraire, l'utilisateur final a toute latitude pour spécifier la structure du document au moment de sa création. Il peut créer de nouveaux types d'éléments selon ses besoins ou sa fantaisie et il les agence à sa guise. Il n'y a pas de structure générique. Ce modèle définit une syntaxe mais n'y associe aucune sémantique, ce qui limite les traitements applicables aux documents. Pratiquement, seule la présentation peut se faire à partir de cette structure, l'utilisateur fournissant les règles de formatage de chacun des types qu'il crée.

Pour Andra, la structure est également relativement libre, puisque le fichier de style associé à un document définit les types d'éléments disponibles et leurs règles de présentation, mais n'indique rien sur la structure logique des documents. L'arbre construit par le système est totalement invisible de l'utilisateur, ce qui donne lieu à des structures parfois curieuses, qui reflètent plus l'historique des commandes d'édition que les intentions de

l'auteur concernant l'organisation de son document. Ce défaut est souligné par les utilisateurs d'Andra [André 84] et a été corrigé dans Lara.

Tous les autres systèmes se fondent sur la même notion de document que Grif : une structure spécifique construite selon le modèle d'une structure logique générique. Cette approche est également celle de [Kimura] dont l'éditeur, très incomplet, n'a pas été retenu dans cette comparaison mais dont le modèle de document est intéressant. Les langages de définition de structure présentent des différences d'ordre syntaxique : alors que la plupart de ces systèmes utilisent une syntaxe de style BNF, Grif propose un langage un peu plus riche, où il est notamment possible de spécifier les cardinalités et d'imposer des valeurs d'attributs.

Bien que la plupart des systèmes utilisent le même modèle, la finesse de la structure qu'ils considèrent varie. Pour certains, le paragraphe constitue un atome (Mentor-Rapport, PEN), alors que d'autres voient différents éléments dans les paragraphes, comme [Hamlet], qui distingue les citations, les parenthèses, etc... Grif permet de définir des structures très fines, même si elles risquent d'être lourdes à manipuler. Il offre en plus une alternative à ce raffinement de la structure : les attributs qui permettent d'ajouter de la sémantique à certaines parties d'éléments de la structure et ils sont plus faciles à manipuler. Cette notion d'attribut se trouve rarement dans les autres systèmes examinés ([Cowan] est la seule exception).

Une autre notion de Grif peu courante dans ces systèmes est la référence. Si elle est parfois mentionnée, c'est le plus souvent dans les systèmes non réalisés ou dans les extensions envisagées des systèmes existants ([Coray], [Walker]). En revanche les autres constructeurs de Grif, et donc les structures arborescentes, se retrouvent dans pratiquement tous les systèmes. Furuta se distingue en traitant en plus les structures tabulaires dans son éditeur spécialisé pour les tableaux. Il est possible que l'absence de ce type de structure dans Grif limite ses possibilités dans le domaine des tableaux complexes, bien qu'une étude mériterait d'être menée sur la représentation arborescente des tableaux.

L'usage qui est fait de la structure dans tous ces systèmes varie beaucoup. Pour certains la structure permet seulement d'associer des règles de présentation aux différents éléments du document (Pléiade, Etude, Andra, Lara). Pour d'autres, elle permet en plus de guider l'utilisateur dans la construction d'un document conforme à un modèle ([Hamlet], [Cowan], W). Quelques systèmes effectuent de plus des numérotations automatiques à partir de la structure ([Coray], [Furuta 86] et Mentor-Rapport). Enfin deux systèmes ajoutent, comme Grif, des traitements plus élaborés, notamment des renvois et des index (PEN, [Walker]).

2.3. L'image des documents

L'image présentée sur l'écran pendant l'édition, comme celle qui est imprimée en fin de traitement, est construite d'après la structure. C'est un principe qui ne souffre pas d'exception dans le domaine des éditeurs structurés. Cependant l'image des documents est considérée de différents points de vue.

Pour certains le principe WYSIWYG est appliqué jusqu'au bout, et chaque document n'a qu'une image, la même sur l'écran d'édition et sur le papier. C'est le cas de Etude, de Lara et de W notamment. D'autres considèrent, comme Grif, que plusieurs images sont possibles pour le même document, et que le changement de présentation doit se faire simplement en changeant les règles de présentation et non pas en modifiant le document. C'est le cas de [Cowan], [Coray], [Furuta 86] et Andra.

La question des pages est traitée d'une façon variée. Le point de vue de Grif n'est pas retenu par tous : certains systèmes affichent des pages à l'écran, ce sont ceux qui lient étroitement l'édition et le formatage, comme [Coray], Pléiade, [Hamlet] ou Etude. D'autres, au contraire donnent une image assez lointaine de ce qui sera imprimée, c'est notamment le cas de Mentor-Rapport et de [Furuta 86]. Grif se situe entre ces deux extrêmes en ignorant les pages, mais en permettant une bonne qualité de l'image affichée.

Le mécanisme des vues de Grif se retrouve dans de nombreux systèmes, avec des variantes (mais pas dans Andra, Lara ni PEN). Certains systèmes prédéfinissent une vue montrant le squelette du document (Pléiade, [Furuta 86], W, Mentor-Rapport, [Walker]), mais peu offrent un mécanisme général permettant de définir l'aspect et le contenu d'un nombre quelconque de vues, comme dans Grif.

Bien que les vues constituent une façon de visualiser la structure des documents, d'autres moyens sont disponibles dans la plupart des systèmes. Certains affichent le chemin dans l'arbre depuis la racine jusqu'à l'élément courant (Pléiade, [Furuta 86]), d'autres écrivent systématiquement le type des éléments dans une zone spécifique, en regard de l'image du document (Etude) ou dans l'image même du document ([Furuta 86]). L'idée des boîtes de présentation de Grif se retrouve dans plusieurs systèmes ([Coray], W, [Furuta 86]), mais, curieusement, cette notion intervient dans la définition de la structure générique, et non pas au niveau des règles de présentation.

Dans tous les systèmes, le concept de boîte est largement utilisé pour la construction des images affichées ou imprimées, mais seul W le rend directement accessible dans les règles de présentation, qui permettent de spécifier les positions relatives des boîtes, avec, semble-t-il, moins de précision que dans Grif. Tous les autres systèmes sont largement influencés par Scribe : les règles de présentation sont exprimées sous la forme d'attributs typographiques conventionnels : corps, style des caractères, marges, retraits, etc...

2.4. Les objets inclus

Pour certains systèmes, le seul contenu possible des documents est le texte ([Hamlet], [Cowan], [Walker], Etude, Lara). Andra autorise l'insertion d'images dans les documents, mais celles-ci ne sont pas traitées par l'éditeur, elles sont seulement insérées au moment de l'impression du document. La plupart des autres systèmes autorisent l'inclusion d'éléments de différentes natures et traitent ces objets avec des outils spécifiques ([Furuta 86], Pléiade, PEN, extension prévue pour [Coray]). Comme Grif, W traite les formules mathématiques de la même façon que le reste de la structure du document, mais seul Grif manipule de façon homogène toutes sortes d'objets, définissables par les utilisateurs, avec les mêmes outils que pour les documents eux-mêmes.

Mentor-Rapport se place d'une façon un peu particulière, puisque, comme Grif, il dispose d'un mécanisme général pour traiter dans le même document plusieurs symbolismes différents, mais ses moyens de présentation (décompilation dans la terminologie Mentor) sont trop limités pour afficher des objets bidimensionnels comme les formules, les tableaux ou les figures. La manipulation des ces types d'objets est d'ailleurs annoncée pour une future version de Mentor-Rapport.

2.5. Le mode d'interaction

L'interface utilisateur reflète d'assez près la source d'inspiration des concepteurs de ces systèmes. Ceux qui subissent le plus directement l'influence des environnements de programmation conservent un dialogue utilisateur semblable à celui des programmeurs, avec des commandes explicites de traitement d'arbres (Mentor-Rapport, [Hamlet], [Walker]). Ce sont d'ailleurs les mêmes qui utilisent un éditeur de texte préexistant (par exemple EMACS [Stallman]), pour la manipulation des atomes de la structure. Les autres sont plus orientés vers des interfaces à base de menus et souris et intègrent plus étroitement le traitement de texte.

Le degré d'interactivité est assez variable d'un système à l'autre. Certains s'efforcent de réfléchir immédiatement sur l'écran toute action de l'utilisateur, y compris la frappe d'un seul caractère de texte (Etude, Pléiade, Andra, Lara), d'autres ne reformatent l'image du document que dans certaines circonstances, telles qu'une commande explicite de l'utilisateur (W) ou la fin de la saisie d'un élément ([Furuta 86], Mentor-Rapport). Grif utilise les deux techniques à la fois, en effectuant un réaffichage au caractère près dans la vue choisie par l'utilisateur, et un réaffichage différé dans les autres vues.

Tous les systèmes ne donnent pas la même importance à l'interface utilisateur. C'est un point fort pour Etude où les aides en ligne et l'annulation de la dernière commande, par exemple, sont disponibles à tout moment, quel que soit le contexte. Pléiade et [Cowan] adoptent également ce point de vue, bien que d'une manière moins systématique. Sur ce thème, Grif ne propose que peu de choses, mais son architecture générale offre toutes les possibilités d'extension du Médiateur et permet ainsi à l'utilisateur d'adapter l'interface à ses besoins.

2.6. Les autres caractéristiques

Deux propositions ([Hamlet] et [Walker]) évoquent l'adjonction à un éditeur structuré des outils d'aide à l'écriture déjà disponibles pour les formateurs : détection de fautes d'orthographe, contrôle du style, dictionnaires de synonymes, etc... Bien que ces outils présentent un intérêt certain, ils ne semble pas qu'ils aient été mis en œuvre dans un éditeur de documents structurés.

Le mode d'impression des documents change d'un système à l'autre. Ceux qui attachent la plus grande attention au formatage en se conformant de près à l'approche WYSIWYG prennent eux-même en charge l'impression des documents ([Coray], Pléiade, W, Etude, Andra, Lara). Cette conception conduit à une architecture relativement complexe pour [Coray], où, lors de l'édition du document, le formatage se fait par parties et en parallèle. Dans Pléiade, certaines concessions sont faites sur la qualité du formatage : les algorithmes utilisés sont le résultat d'un compromis entre l'efficacité et la qualité du résultat obtenu. Mais la plupart des systèmes utilisent, comme Grif, les formateurs disponibles. [Cowan] propose un mécanisme général, comparable à celui de Grif, donnant ainsi l'accès à tout formateur. [Walker] utilise Scribe. Mentor-Rapport offre le choix entre Compose, Troff et TEX. W doit, à terme, utiliser TEX. [Furuta 86] vise TEX, Scribe et Troff.

Aucun de ces systèmes ne se conforme à une norme de représentation de documents. Le modèle de document qu'ils utilisent pour la plupart n'est pas très éloigné de celui de ODA [Horak] ou de SGML [ISO]. Cependant les systèmes qui, comme Pléiade, Interleaf ou Andra, n'ont pas de structure logique générique n'utiliseront pas toutes les possibilités offertes par les standards. Inversement, il n'est pas sûr qu'un standard comme ODA puisse exprimer toutes les structures produites par ces systèmes, comme notamment les références ou les structures de tableaux ou de formules mathématiques : les architectures de contenu de ODA sont encore trop limitées.

3. LES POTENTIALITES DE L'EDITION STRUCTUREE

Dans tous les systèmes que nous venons de voir, la structure logique spécifique des documents est fournie explicitement par l'utilisateur. Celui-ci est le plus souvent aidé dans cette tâche par les structures génériques, mais c'est néanmoins lui qui indique, par des commandes appropriées, les différents éléments structurels constituant le document. La seule exception est proposée par [Hamlet], qui suggère, par exemple, que le système effectue lui-même les changements de paragraphes selon le contenu. Encore ne s'agit-il que d'une proposition.

Si on se réfère au domaine de l'édition syntaxique des programmes, les éditeurs de documents marquent de ce point de vue un certain retard. Dans un environnement de programmation, il est souvent possible d'entrer des fragments de programme sous une forme textuelle, qui est ensuite analysée pour en extraire la structure. Dans les documents, cette structure est considérée à un niveau moins fin : on ne décompose pas une phrase ou un paragraphe de la même façon qu'une expression arithmétique dans un programme. La nécessité de l'analyse se fait donc sentir moins fortement, mais il est certain que les éditeurs de documents devraient ajouter des fonctions d'analyse à la synthèse qu'ils effectuent actuellement.

Deux tendances se dessinent dans l'analyse structurelle des documents. L'une touche au domaine de la linguistique, et cherche à retrouver l'organisation du discours à partir de son contenu. L'autre est plus proche de la reconnaissance des formes, et propose de reconstituer la structure logique des documents à partir de leur mise en page. La première peut être vue comme une aide à la création des documents : l'auteur écrit et ne se préoccupe pas des détails de structuration logique de son texte. La deuxième serait surtout utile pour faire entrer dans un système d'édition structurée des documents qui ont été produits par d'autres systèmes et qui ne sont disponibles que sous une forme imprimée ou sous une forme électronique, mais purement physique.

On peut se référer aux éditeurs syntaxiques de programmes également pour évaluer les chances d'acceptation de l'édition structurée des documents. Bien que les programmeurs aient maintenant des éditeurs syntaxiques à leur disposition, nombreux sont ceux qui utilisent encore des éditeurs de texte conventionnels. On peut penser cependant que l'évolution se fera plus vite dans le domaine de l'édition des documents et pour deux raisons principalement.

La première raison est l'évolution progressive et continue de la gamme des outils, qui vont du simple éditeur de texte au système le plus sophistiqué. Il n'y a pas de seuil à franchir pour passer d'une machine de traitement de texte à un système purement WYSIWYG (Macwrite par exemple), puis à un système à structure faible (comme

Interleaf) et à un système plus fortement structuré (Grif). Bien que les fonctionnalités s'enrichissent, le mode d'utilisation n'est pas profondément différent.

La deuxième raison est le poids que commencent à prendre les normes de représentation de documents. SGML et ODA se fondent sur un concept de document structuré logiquement, les documents représentés selon ces normes se prêtent donc bien à l'édition structurée. Si, à terme, le seul moyen efficace d'échanger des documents électroniques éditables est de les représenter selon ces normes, la technique de l'édition structurée s'imposera naturellement. Cependant, dans un premier temps, les normes sont plutôt vues comme des représentations d'échange, les documents étant produits avec les systèmes existants et convertis par des passerelles. Il n'existe pas encore d'éditeur interactif qui mette en œuvre les modèles d'ODA ou de SGML, et il semble qu'il faudra encore attendre pour disposer d'outils conçus spécifiquement pour produire des documents conforme à ODA. Grif, ou d'autres systèmes présentés dans ce chapitre, pourraient servir de base pour la construction de tels éditeurs.

ANNEXE 1

LE META-LANGAGE M

On présente ici la grammaire du méta-langage M utilisé dans la description des langages de Grif. Ce méta-langage permet, dans les chapitres précédents, de présenter les langages S et P. Il est également utilisé par les compilateurs de S et P. Ce langage se décrit ici lui-même :

```
{ Les textes entre accolades sont des commentaires }
Grammaire =     Regle < Regle > 'END' .
                { Les signes < et > indiquent une répétition
                (zéro, une ou plusieurs fois) }
                { END marque la fin de la grammaire }
Regle =         Ident '=' PartieDroite '.' .
                { Le point indique la fin d'une règle }
PartieDroite =  DrTerminal / DrIntermed .
                { La barre oblique indique une alternative }
DrTerminal =    'NAME' / 'STRING' / 'NUMBER' .
                { Partie droite d'une règle terminale }
DrIntermed =    Possibilite < '/' Possibilite > .
                { Partie droite d'une règle intermédiaire }
Possibilite =   ElRepetOpt < ElRepetOpt > .
ElRepetOpt =    Element / '[' Element < Element > ']' /
                '<' Element < Element > '>'  .
                { Les crochets délimitent une partie optionelle }
Element =       Ident / MotCle .
Ident =         NAME .
                { Identificateur, suite de caractères }
MotCle =        STRING .
                { Chaîne de caractères délimitée par des apostrophes }
END
```

ANNEXE 2

LE LANGAGE S

Cette annexe rassemble toutes les règles de la grammaire du langage S, qui est présentée dans le chapitre 3.

```
{ Définition d'un schéma de structure }
SchemaStruct = 'STRUCTURE' IdentElem ';'  { Nom de la structure générique }
               'DEFPRES' IdentPres ';'    { Présentation par défaut }
             [ 'ATTR' SuiteAttr ]         { Définition des attributs }
             [ 'PARAM' SuiteRegles ]      { Définition des paramètres }
               'STRUCT' SuiteRegles       { Règles de structuration }
             [ 'ASSOC' SuiteRegles ]      { Définition des éléments associés }
             [ 'UNITS' SuiteRegles ]      { Unités exportées }
               'END' .

IdentElem = NAME .          { Nom d'un type d'élément }
IdentPres = NAME .          { Nom d'un schéma de présentation }

SuiteAttr = Attribut < Attribut > .
{ Définition d'un attribut }
Attribut =  IdentAttr '=' '(' ValAttr < ',' ValAttr > ')' ';' .
IdentAttr = NAME .          { Nom d'un attribut }
ValAttr =   NAME .          { Une valeur d'un attribut }

SuiteRegles =    Regle ';' < Regle ';' > .
{ Définition d'une règle de structuration }
Regle =          IdentElem '=' DefAvecAttr .
DefAvecAttr =    Definition [ 'WITH' SuiteAttrFixes ] .
{ Liste des attributs à valeur imposée pour un type d'élément }
SuiteAttrFixes = AttrFixe < ',' AttrFixe > .
AttrFixe =       IdentAttr '=' ValAttr .

Definition = TypeDeBase / Constr / Element .
```

```
{ Les quatres types de base, plus le type indéterminé UNIT }
TypeDeBase = 'TEXT' / 'GRAPHICS' / 'SYMBOL' / 'PICTURE' / 'UNIT' .
{ Types de mêmes structures }
Element =    IdentElem [ '=' Definition ] .

{ Définition des types construits }
Constr = 'LIST' [ '[' min '..' max ']' ] 'OF' '(' DefAvecAttr ')' /
         'BEGIN' SuiteDef 'END' /          { Agrégat }
         'CASE' 'OF' SuiteDef 'END' /      { Choix }
         'REFERENCE' '(' IdentElem ')' .   { Référence }

min =      Entier / '*' .      { Nombre minimum d'éléments d'une liste }
max =      Entier / '*' .      { Nombre maximum d'éléments d'une liste }
Entier =   NUMBER .            { Un entier sans signe }
SuiteDef = DefAvecAttr ';' < DefAvecAttr ';' > .

END                            { Fin de la grammaire }
```

ANNEXE 3

LE LANGAGE P

Cette annexe rassemble toutes les règles de la grammaire du langage P, qui est présentée de façon détaillé dans le chapitre 3.

```
{ Définition d'un schéma de présentation }
SchemaPres = 'PRESENTATION' IdentType ';' { Structure générique }
           [ 'VIEWS' SuiteVues ]          { Déclaration des vues }
           [ 'COUNTERS' SuiteCompteurs ]  { Déclaration des compteurs }
           [ 'CONST' SuiteConst ]         { Déclaration des constantes }
           [ 'VAR' SuiteVar ]             { Déclaration des variables }
             'DEFAULT' SuiteReglesVues    { Règles par défaut }
           [ 'BOXES' SuiteBoites ]        { Boîtes de présentation }
             'RULES' SuitePresent         { Règles associées aux types }
           [ 'ATTRIBUTES' SuitePresAttr ] { Règles associées aux attributs }
             'END' .

IdentType = NAME .          { Un type défini dans la structure générique }

SuiteVues = IdentVue < ',' IdentVue > ';' .
IdentVue =  NAME .          { Nom d'une vue }

SuiteCompteurs = Compteur < Compteur > .
Compteur =       IdentCompteur ':' FonctCompteur ';' .
IdentCompteur =  NAME .     { Nom d'un compteur de numérotation }
{ Définition d'une fonction de comptage }
FonctCompteur = 'RANK' 'OF' IdentType /          { Rang dans une liste }
                'SET' ValeurCompt 'ON' IdentType { Initialisation }
                'ADD' ValeurCompt 'ON' IdentType . { Incrément }
ValeurCompt =    NUMBER . { Valeur d'initialisation ou d'incrément }

SuiteConst =  Const < Const > .
{ Définition d'une constante }
```

```
Const =        IdentConst '=' TypeConst ValeurConst ';' .
IdentConst =   NAME .          { Nom d'une constante }
TypeConst =   'Text' / 'Symbol' / 'Graphics' / 'Picture' .
ValeurConst = STRING .         { Valeur d'une constante }

SuiteVar =         Variable < Variable > .
{ Définition d'une variable }
Variable =         IdentVar ':' SuiteFonctions ';' .
IdentVar =         NAME .      { Nom d'une variable }
SuiteFonctions = Fonction < Fonction > .
{ Une fonction donnant une valeur à une variable }
Fonction =       'DATE' / 'FDATE' /
                  IdentConst / TypeConst ValeurConst /
                 'VALUE' '(' IdentCompteur ',' StyleCompteur ')' .
{ Style d'affichage de la valeur d'un compteur de numérotation }
StyleCompteur = 'Arabic' / 'Roman' / 'Alphabetic' .

SuiteBoites = Boite < Boite > .
{ Définition d'une boîte de présentation ou de mise en page }
Boite =       'FORWARD' IdentBoite ';' /
               IdentBoite ':' SuiteReglesVues .
IdentBoite =  NAME .    { Nom d'une boîte de présentation ou de mise en page }

SuitePresent = Present < Present > .
{ Les règles de présentation attachées à un type d'élément }
Present =       IdentType ':' SuiteReglesVues .

SuitePresAttr = PresAttr < PresAttr > .
{ Les règles de présentation attachées à la valeur d'un attribut }
PresAttr =      IdentAttr '=' ValeurAttr ':' SuiteReglesVues .
IdentAttr =     NAME .    { Nom d'un attribut du schéma de structure }
ValeurAttr =    NAME .    { Valeur d'un attribut du schéma de structure }

{ Une suite de règles de présentation pour les différentes vues }
SuiteReglesVues = 'BEGIN' < Regle > < ReglesVue > 'END' ';' /
                   ReglesVue / Regle .
{ Suite des règles de présentation spécifiques à une vue }
ReglesVue =       'IN' IdentVue SuiteRegles .
SuiteRegles =     'BEGIN' Regle < Regle > 'END' ';' / Regle .
```

```
Regle =   Reglel ';' / Regle2 ';' .
{ Une règle de présentation }
Reglel = 'VertRef' ':' PositionHoriz / { Position de l'axe de référ. vertical }
         'HorizRef' ':' PositionVert / { Position de l'axe de référ. horizontal }
         'Height' ':' Dimension /      { Hauteur de la boîte }
         'Width' ':' Dimension /       { Largeur de la boîte }
         'VertPos' ':' PosH /          { Position verticale }
         'HorizPos' ':' PosV /         { Position horizontale }
         'Expand' ':' Booleen /        { Extensibilité de la boîte }
         'Break' ':' Booleen /         { Sécabilité de la boîte }
         'Size'  ':' NombreHerit /     { Facteur de taille du contenu }
         'Visibility' ':' NombreHerit /{ Degré de visibilité de la boîte }
         'HighLight' ':' NombreHerit / { Degré de mise en évidence du contenu}
         'Indent' ':' DistanceHerit /  { Retrait de la 1ère ligne contenue }
         'Content' ':' VarConst .      { Contenu de la boîte }
{ Une fonction de présentation }
Regle2 =  Creation '(' IdentBoite ')' /        { Création d'une boîte }
          MiseEnPage [ '(' IdentBoite ')' ] / { Mode de mise en page }
         'Copy' '(' IdentBoite ')' .          { Copie du contenu d'une boîte }

{ Un axe ou un côté horizontal d'une boîte }
AxeHoriz = 'Top' / 'HMiddle' / 'HRef' / 'Bottom' .
{ Un axe ou un côté vertical d'une boîte }
AxeVert =  'Left' / 'VMiddle' / 'VRef' / 'Right' .

{ Définition de la position d'un axe horizontal }
PosH = 'nil' / AxeHoriz '=' PositionVert .
PositionVert =  Reference '.' AxeHoriz [ Distance ] .
{ Définition de la position d'un axe vertical }
PosV = 'nil' / AxeVert '=' PositionHoriz .
PositionHoriz = Reference '.' AxeVert [ Distance ] .

{ Définition d'une boîte de référence pour une position ou une dimension }
Reference = 'Enclosing' /                 { La boîte englobante }
            'Enclosed' [ NBoiteType ] / { Une boîte englobée }
            'Previous' [ NBoiteType ] / { Une boîte sœur précédente }
            'Next' [ NBoiteType ] /     { Une boîte sœur suivante }
            '*' /                       { La boîte elle-même }
             BoiteType .                { La boîte sœur d'un type donné }
{ Un type de boîte d'élément structuré, de présentation ou de mise en page }
```

```
BoiteType =  IdentBoite / IdentType .
NBoiteType = [ 'NOT' ] BoiteType .

{ Distance entre deux axes ou côtés de boîtes }
Distance =    [ '+' ]  DistancePos / '-' DistanceNeg .
DistancePos = NUMBER .
DistanceNeg = NUMBER .

{ Hauteur ou largeur d'une boîte }
Dimension =    Reference '.' HautLarg [ Rapport ] /
               DimensionAbs .
HautLarg =    'Height' / 'Width' .
{ Pourcentage ou différence entre les hauteurs ou largeurs de deux boîtes }
Rapport =     '*' RapportDim '%' / Distance .
RapportDim =  NUMBER .       { Pourcentage }
DimensionAbs = NUMBER .      { Dimension absolue }

{ Règle d'héritage }
Heritage =     Parente  ValeurHerit .
{ lien de parenté de l'élément qui transmet l'héritage }
Parente =     'Enclosing' / 'Enclosed' / 'Previous' .
{ Différence de la valeur héritée par rapport à celle de l'élément qui transmet l'héritage}
ValeurHerit = '+' EntierPos [ 'Max' maximum ] /
              '-' EntierNeg [ 'Min' minimum ] /
              '=' .
EntierPos =    NUMBER .
EntierNeg =    NUMBER .
maximum =      NUMBER .      { Valeur maximum d'une valeur héritée }
minimum =      NUMBER .      { Valeur minimum d'une valeur héritée }

NombreHerit = Entier / Heritage .
Entier =      NUMBER .       { Un entier non signé }

Booleen =  'Yes' / 'No' .    { Une valeur booléenne }

DistanceHerit = Distance / Heritage .

{ Mode de création d'une boîte de présentation }
Creation =    [ 'IF' Condition ] Cree.
{ Condition de création d'une boîte de présentation }
```

```
Condition =  [ 'NOT' ] FirstLast .
FirstLast = 'First' / 'Last' .
```
{ Position structurelle de la boîte de présentation créée }
```
Cree =       'Create' / 'CreateBefore' / 'CreateAfter' .
```
{ Fonction de mise en page du contenu d'une boîte }
```
MiseEnPage = 'Line' / 'Column' / 'Page' .

END
```
{ Fin de la grammaire }

Dans le langage P, comme dans la représentation pivot des documents Grif, les éléments de base graphiques et symboles sont codés de la façon suivante :

Graphique :

'C' un cercle inscrit dans la boîte : ○,
'R' un rectangle qui est le contour de la boîte : □,
't' un trait horizontal : le côté supérieur de la boîte : ¯,
'h' un trait horizontal au milieu de la boîte, de la longueur de la boîte : −,
'b' un trait horizontal : le côté inférieur de la boîte : _,
'l' un trait vertical : le côté gauche de la boîte : |,
'v' un trait vertical au milieu de la boîte, de la hauteur de la boîte : |,
'r' un trait vertical : le côté droit de la boîte : |,
' ' un élément transparent : .

Symbole:

'r' un radical de la taille de la boîte : √,
'i' une intégrale simple de la hauteur de la boîte : ∫,
'd' une intégrale double de la hauteur de la boîte : ∫∫,
't' une intégrale triple de la hauteur de la boîte : ∫∫∫,
'S' le symbole de la sommation, de la taille de la boîte : Σ,
'P' le symbole du produit de la taille de la boîte : Π,
'U' le symbole de l'union de la taille de la boîte : ∪,
'I' le symbole de l'intersection de la taille de la boîte : ∩,
'>' une flèche à droite, de la longueur de la boîte : →,
'<' une flèche à gauche, de la longueur de la boîte : ←,
'^' une flèche vers le haut, de la hauteur de la boîte : ↑,
'_' une flèche vers le bas, de la hauteur de la boîte : ↓.

ANNEXE 4

LA REPRESENTATION PIVOT

Les principes qui ont guidé la conception de la représentation pivot des documents de Grif sont exposés dans le chapitre 4. On trouvera ici le détail de cette représentation.

Rappelons qu'un schéma de structure, une fois compilé, est essentiellement constitué de deux tables :

- la table des attributs,
- la table des types d'éléments.

Un attribut porte un nom et a un nombre limité de valeurs possibles. Chacune de ces valeurs a un numéro d'ordre et un nom. La valeur d'un attribut peut être désignée par ce numéro d'ordre. Un attribut peut être désigné par son rang dans la table des attributs (ce rang est également appelé numéro de l'attribut).

Tous les types d'éléments définis par une structure générique sont décrits dans la table des types d'éléments. Cette table contient, dans l'ordre, les types de base (chaîne de caractères, symbole mathématique, élément graphique, image...), puis les constantes, puis tous les éléments structurés, associés et paramètres. Quelle que soit sa catégorie, un type d'élément peut être repéré par son rang dans cette table (le rang est également appelé numéro du type). Ce rang indique la catégorie de l'élément : les premiers numéros sont ceux des types de base, les suivants sont ceux des éléments structurés.

Dans la table des types on trouve des constantes. Comme tous les autres types d'éléments, une constante porte un nom. De plus, elle a une valeur définie dans le schéma de structure : une chaîne de caractères.

La notation utilisée pour décrire la représentation pivot est celle qui a déjà été utilisée pour décrire les langages de Grif. On y ajoute les conventions suivantes :

`CHAINECAR` représente une chaîne de caractères ASCII imprimables terminée par un octet nul.

`CARACTERE` représente un seul caractère ASCII imprimable, c'est à dire un caractère dont le code est supérieur ou égal à 32 (décimal).

NUMERO représente un entier positif codé sur deux octets.

0 suivi d'un ou deux chiffres représente un octet ayant pour valeur le nombre décimal représenté par le (ou les) chiffre(s) qui suivent.

La représentation pivot d'un document contient d'abord une chaîne de caractères qui est le nom de la classe du document. Ce nom se termine par un octet nul. Lorsque l'éditeur doit travailler sur un document, il trouve ainsi le schéma de structure qu'il doit charger et qui le guidera dans l'édition du document.

Si le document comporte des objets de différentes natures, les classes de ces objets sont listées juste après le nom de la classe du document. Chaque classe est représentée par un octet `Marque-Classe` suivi du nom de la classe. Toutes les classes d'objets utilisées dans le document (et seulement celles qui sont réellement utilisées dans ce document) sont indiquées ainsi, qu'elles soient explicitement référencées par le schéma de structure du document ou qu'elles aient été appelées grâce à une règle `UNIT`. Le rang d'une classe dans cette liste est appelé numéro de classe. On verra plus loin qu'il sert à indiquer dans quel schéma de structure est défini un attribut, ou a quelle classe appartient un objet.

Si certains paramètres ont une valeur dans le document, ces valeurs de paramètres suivent les noms des classes d'objets, ou le nom de la classe du document s'il n'y a pas de noms de classes d'objets. Chaque valeur de paramètre est précédée d'un octet `Marque-Param` indiquant qu'il s'agit d'une valeur de paramètre. Les valeurs de paramètres sont représentées comme tout élément structuré(voir plus loin). Pour un document, les paramètres possibles, définis dans le schéma de structure, n'ont pas tous une valeur. Seuls ceux qui ont une valeur apparaissent ainsi en tête de la représentation pivot du document. Si aucun paramètre n'a de valeur, ou si aucun paramètre n'est défini dans le schéma de structure du document, aucune valeur de paramètre n'est présente dans la représentation pivot. En revanche, si un paramètre a une valeur, il ne peut en avoir qu'une et donc toutes les valeurs de paramètres d'un document sont de types différents.

Après les valeurs des paramètres viennent les éléments associés du document (ces éléments peuvent être absents). Les éléments associés sont groupés par type : un document peut avoir plusieurs éléments associés de même type. Un octet `Marque-Associé` précède chaque suite d'éléments du même type. Dans ces suites, les éléments associés sont représentés comme tout élément structuré.

Après les éléments associés vient le document lui-même. Il est précédé d'un octet `Marque-Document` et suivi d'un octet `Marque-Fin-Document`. Le document est représenté comme tout élément structuré.

```
Représ-Pivot =    Classe-Document
                < Marque-Classe Nom-Classe >
                < Marque-Param Param >
                < Marque-Associé Suite-Associés >
                  Marque-Document  Le-Document  Marque-Fin-Document .

Classe_Document = CHAINECAR .

Marque-Classe =   O14 .
Nom-Classe =      CHAINECAR .

Marque-Param =    O16 .
Param =           Elément .

Marque-Associé =  O2 .
Suite-Associés =  Elément < Elément > .

Marque-Document = O19 .
Le-Document =     Elément .
Marque-Fin-Document = O26 .
```

Les éléments structurés (paramètres, éléments associés, éléments du document) sont tous représentés de la même façon.

Chaque élément commence par une indication de son type. Le type est indiqué par un octet `Marque-Type` suivi du numéro de type de l'élément (un entier positif sur deux octets). Dans le cas où le type de l'élément est défini dans un schéma de structure différent de celui où est défini le type de l'élément dont il est le descendant direct, le nouveau schéma de structure est indiqué, avant la marque de type et le numéro de type, par un octet `Marque-Classe` suivi du numéro de la nouvelle classe d'objets.

Après l'indication de type, et si l'élément est référencé, apparaît un label, qui est constitué d'un octet `Marque-Label` et d'un entier positif (sur deux octets) qui identifie l'élément de façon unique dans le document. C'est cet entier qui sera utilisé dans les références qui pointent sur cet élément.

Chaque élément peut être affecté d'un ou plusieurs attributs, qui portent sur l'élément lui-même et tout son contenu. Seuls sont indiqués les attributs qui ne sont pas imposés dans le schéma de structure par une instruction `WITH`. Chaque indication d'attribut comprend, dans l'ordre :

- un octet Marque-Attribut,
- le numéro de la classe où l'attribut est défini : zéro pour la classe du document ou le rang d'une classe d'objets dans la liste des classes figurant en tête de la représentation pivot,
- le numéro de l'attribut dans le schéma de structure de la classe où il est défini,
- le numéro de la valeur de cet attribut affectée à l'élément.

Enfin vient le contenu de l'élément. Ce contenu est absent pour un élément vide.

```
Elément =          Type-Elément
                 [ Marque-Label Numéro-Label ]
                 < Marque-Attribut  Numéro-Classe  Numéro-Attribut
                   Numéro-Valeur >
                 [ Contenu ] .

Type-Elément =     [ Marque-Classe Numéro-Classe ]
                   Marque-Type Numéro-Type.
Numéro-Classe =    NUMERO .
Marque-Type =      O20 .
Numéro-Type =      NUMERO .

Marque-Label =     O12 .
Numéro-Label =     NUMERO .

Marque-Attribut =  O1 .
Numéro-Attribut =  NUMERO .
Numéro-Valeur =    NUMERO .
```

La représentation du contenu des éléments diffère selon que l'élément représenté est une feuille de la structure, une référence, un élément structuré, un paramètre ou une constante. Le schéma de structure permet d'interpréter correctement la représentation du contenu, puisque la structure de l'élément y est définie.

S'il s'agit d'une feuille, le contenu est simplement constitué d'un octet Marque-Début suivi du contenu de la feuille et d'un octet Marque-Fin. Si la feuille est vide, le contenu est réduit à un octet nul.

Si la feuille est du type chaîne de caractères ou image (c'est indiqué dans le schéma de structure), le contenu de la feuille est une suite de caractères terminée par un octet nul. S'il

s'agit d'un élément graphique ou d'un symbole mathématique, le contenu est réduit à un seul caractère représentant l'élément graphique ou le symbole.

Dans le cas d'une chaîne de caractères, le contenu est simplement la suite des codes ASCII des caractères qui constituent la chaîne. Dans le cas d'une image, ce n'est pas l'image elle-même qui constitue le contenu de la feuille, mais le nom du fichier qui la contient.

Si la feuille est du type référence, elle est représentée par un octet Marque-Référ suivi du numéro de label de l'élément référencé (sur deux octets). Notons que l'élément référencé peut se trouver aussi bien avant qu'après la référence. La référence peut aussi ne pas être affectée ; dans ce cas le numéro de label est nul (sur deux octets).

S'il s'agit d'une constante, il n'y a pas de contenu, puisque le numéro de type permet de retrouver, dans le schéma de structure, la valeur de la constante utilisée.

De même, s'il s'agit de l'utilisation d'un paramètre dans le document (et non de la définition de sa valeur), il n'y a pas de contenu, puisque la valeur, s'il y en a une, est unique et a déjà été indiquée au début de la représentation pivot.

S'il s'agit d'un élément structuré, le contenu est formé d'un octet Marque-Début, suivi de la suite des représentations de chacun des éléments qu'il contient au niveau immédiatement inférieur, et d'un octet Marque-Fin. Si cet élément est vide, son contenu est simplement marqué par les deux octets Marque-Début et Marque-Fin.

```
Contenu =          Feuille / Référence / Constante /
                   Paramètre / Elem-Struct .

Feuille =          Marque-Début Contenu-Feuille Marque-Fin .
Contenu-Feuille =  CHAINECAR / CARACTERE .
Marque-Début =     O4 .
Marque-Fin =       O6 .

Référence =        Marque-Référ  Numéro-Label .

Marque-Référ =     O18 .

Constante =        NIL .
Paramètre =        NIL .

Elem-Struct =      Marque-Début  Suite-Elem  Marque-Fin .
Suite-Elem =     < Elément > .
```

Pour récapituler :

```
{ La représentation pivot d'un document }
Représ-Pivot =      Classe-Document   { Nom du schéma de structure du document }
                    { Noms des classes des objets contenus dans le document }
                  < Marque-Classe Nom-Classe >
                    { Valeur des paramètres pour le document }
                  < Marque-Param Param >
                    { Les éléments associés du document }
                  < Marque-Associé Suite-Associés >
                    { Le document lui-même }
                    Marque-Document  Le-Document  Marque-Fin-Document .

Classe-Document = CHAINECAR . { Nom du schéma de structure du document }

Nom-Classe =      CHAINECAR . { Nom d'une classe d'objets }

Param =           Elément .   { La valeur d'un paramètre }

{ Tous les éléments associés d'un même type }
Suite-Associés =  Elément < Elément > .

Le-Document =     Elément .

{ Un élément structuré }
Elément =           Type-Elément                  { Type de l'élément }
                  [ Marque-Label Numéro-Label ]   { Label }
                  < Marque-Attribut  Numéro-Classe Numéro-Attribut
                    Numéro-Valeur >               { Attributs de l'élément }
                  [ Contenu ] .                   { Contenu de l'élément }

Type-Elément =    [ Marque-Classe Numéro-Classe ]
                  Marque-Type Numéro-Type .

Contenu =         Feuille / Référence / Constante /
                  Paramètre / Elem-Struct .

{ Contenu d'un élément feuille }
Feuille =         Marque-Début Contenu-Feuille Marque-Fin .
Contenu-Feuille = CHAINECAR / CARACTERE .
```

{ Contenu d'un élément référence }

```
Référence =        Marque-Référ Numéro-Label . { Label de l'élément référencé }

Constante =        NIL .        { Une constante n'a pas de contenu }
Paramètre =        NIL .        { Un paramètre n'a pas de contenu }
```

{ Contenu d'un élément structuré }

```
Elem-Struct =      Marque-Début  Suite-Elem  Marque-Fin .
Suite-Elem =     < Elément > .
```

{ Tous les numéros }

```
Numéro-Type =      NUMERO .
Numéro-Label =     NUMERO .
Numéro-Classe =    NUMERO .
Numéro-Attribut =  NUMERO .
Numéro-Valeur =    NUMERO .
```

{ Toutes les marques }

```
Marque-Classe =    O14 .
Marque-Param =     O16 .
Marque-Associé =   O2 .
Marque-Document =  O19 .
Marque-Fin-Document = O26 .
Marque-Type =      O20 .
Marque-Label =     O12 .
Marque-Attribut =  O1 .
Marque-Début =     O4 .
Marque-Fin =       O6 .
Marque-Référ =     O18.

END                                { Fin de la grammaire }
```

ANNEXE 5

DONNEES TECHNIQUES

1. EDIMATH

Edimath est écrit en Pascal. L'éditeur interactif est composé de quatre modules et le formateur d'un seul. Dans toutes les versions le découpage en modules est le même. On donne ici la taille de ces modules dans la version VAX Unix, qui gére un terminal alpha-numérique.

Un module de l'éditeur prend en charge la saisie des expressions mathématiques, qu'il s'agisse de la création d'une nouvelle formule ou d'ajouts dans une formule existante. Ce module contient également les initialisations et la boucle principale de l'éditeur.

Un autre module s'occupe de la construction des boîtes. C'est lui qui applique les règles de construction des boîtes et calcule les modifications à apporter à l'image affichée.

Le troisième module traite toutes les commandes de sélection et d'édition.

Le dernier module de l'éditeur gére la saisie et l'affichage. Il décode les touches de fonction et envoie les commandes qui tracent la formule sur l'écran à partir de sa représentation sous forme de boîtes.

L'ensemble de l'éditeur, dans la version VAX Unix, représente un total de 5479 lignes de Pascal, y compris les déclarations globales. Lors de l'exécution, il occupe environ 110 Koctets en mémoire.

Comme pour l'éditeur, plusieurs versions du formateur ont été développées, pour des classes d'imprimantes différentes. Une version est notamment utilisée avec les imprimantes alpha-numériques, une autre avec les imprimantes à laser. C'est cette dernière qui a été intégrée dans le formatteur Sroff.

Le tableau suivant indique le volume de chaque partie :

Saisie des expressions mathématiques..... 1 723 lignes
Construction des boîtes..... 1 388 lignes
Commandes d'édition et de sélection..... 1 126 lignes
Saisie et affichage..... 1 087 lignes

Total..... 5 324 lignes

Formateur pour imprimante alpha-numérique..... 1 370 lignes
Formateur pour imprimante laser..... 1 549 lignes

2. GRIF

Grif se compose de quatre grandes parties : les compilateurs, l'Editeur, le Médiateur et le Traducteur. Tous ces programmes sont écrits en Pascal, à l'exception de quelques parties spécifiques en C. Ils travaillent sur des structures de données communes dont les déclarations (constantes et types) représentent environ 1200 lignes de Pascal. Les chiffres donnés ici concernent la première version de Grif, pour une machine Perq sous le système PNX, un Unix doté d'un gestionnaire de fenêtres.

2.1. Les compilateurs

Tous les compilateurs de Grif utilisent le même code d'analyse lexicale et syntaxique. Chaque compilateur comporte de plus une partie spécifique qui engendre des tables utilisées ensuite par l'Editeur ou le Traducteur.

Analyse lexicale et syntaxique..... 867 lignes
Méta-compilateur (langage M)..... 791 lignes
Compilateur du langage S..... 1 414 lignes
Compilateur du langage P..... 2 015 lignes
Compilateur du langage F..... 1 352 lignes

Total..... 6 439 lignes

2.2. L'Editeur

La partie Editeur de Grif se compose de 17 modules, tous écrits en Pascal.

Gestion des documents et des vues..... 1 162 lignes
Chargement et libération des schémas..... 1 066 lignes
Lecture et écriture de la forme pivot..... 944 lignes
Manipulation des arbres abstraits..... 1 787 lignes
Création de nouveaux éléments..... 1 968 lignes
Commandes Copier Couper Coller..... 1 259 lignes
Commandes de recherche..... 994 lignes
Traitement des attributs..... 823 lignes
Gestion de la sélection structurelle..... 906 lignes
Gestion des numérotations..... 410 lignes
Création des images abstraites..... 2 089 lignes
Mise a jour des images abstraites..... 1 489 lignes
Gestion des images partielles..... 976 lignes
Messages et menus en français..... 201 lignes
Messages et menus en anglais..... 201 lignes
Allocation mémoire de l'Editeur..... 487 lignes
"Dumps" (pour la mise au point)..... 436 lignes

Total..... 18 137 lignes

2.3. Le Médiateur

Le Médiateur de Grif se compose de 15 modules en Pascal :

Affichage des boîtes..... 1 348 lignes
Gestion des catalogues..... 589 lignes
Edition du contenu des boîtes..... 1 220 lignes
Défilement dans les fenêtres..... 317 lignes
Mise à jour des images concrètes..... 1 242 lignes
Gestion de la désignation..... 268 lignes
Fonctions de dialogue utilisateur..... 810 lignes
Initialisation et messages du Médiateur..... 286 lignes
Création des images concrètes..... 1 372 lignes
Mise en lignes..... 1 521 lignes
"Dumps" (pour la mise au point)..... 528 lignes
Gestion mémoire du Médiateur..... 408 lignes
Paramétrage du Médiateur..... 139 lignes
Gestion des relations entre boîtes..... 1 097 lignes
Sélection des caractères et des boîtes..... 1 137 lignes

Total..... 12 282 lignes

Le Médiateur comporte aussi trois modules en C, pour l'accès au système d'exploitation et au gestionnaire de fenêtres :

Entrées-sorties..... 621 lignes
Affichage des menus..... 787 lignes
Gestion des fenêtres et de la souris..... 956 lignes

Total..... 2 364 lignes

2.4. Le Traducteur

Le Traducteur, qui adapte un document produit par Grif au langage d'entrée d'un formateur, est constitué d'un seul module :

Traducteur..... 2 234 lignes

REFERENCES BIBLIOGRAPHIQUES

[AAP] Association of American Publishers, *Standard for Electronic Manuscript Preparation and Markup*, A.A.P., Washington, February 1986.

[Achugbue] J. O. Achugbue, "On the line breaking problem in text formatting", *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, Portland, June 1981, et *ACM SIGPLAN Notices*, vol. 16, num. 6, 1981, pp. 117-122.

[Adobe] Adobe Systems Incorporated, *PostScript Language Reference Manual*, Addison-Wesley, Reading, Massachusetts, 1985.

[Agostini] M. Agostini, V. Matano, M. Schaerf, M. Vascotto, "An Interactive User-Friendly TEX in VM/CMS Environment", *TEX for Scientific Documentation*, D. Lucarella ed., Addison-Wesley, 1985, pp. 117-132.

[Aiello] L. Aiello, S. Pavan, "Towards a friendly workstation for TEX", *Internationaler Kongress für Datenverarbeitung um Informations-technologie*, Berlin, 1982.

[Allen] T. Allen, R. Nix, A. Perlis, "PEN: A Hierarchical Document Editor", *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, Portland, June 1981, et *ACM SIGPLAN Notices*, vol. 16, num. 6, 1981, pp. 74-81.

[André 82] J. André, "Bibliographie analytique sur les manipulations de textes", *T.S.I*, vol. 1, num. 5, 1982, pp. 445-455.

[André 84] J. André, R. Ingold, "High Quality Book Production using a Text Editor: A Case Study", *An Introduction to Text Processing Systems*, edited by J. J. H. Miller, Boole Press, Dublin, 1984, pp. 13-20.

[André 85a] J. André, R. Southall, "Experiments in teaching Metafont", *TEX for Scientific Documentation*, D. Lucarella ed., Addison-Wesley, 1985, pp. 141-153.

[André 85b] J. André, Y. Gründt, V. Quint, "Towards an interactive math mode in TEX", *TEX for Scientific Documentation*, D. Lucarella ed., Addison-Wesley, 1985, pp. 79-92.

[André 86] J. André, "GRIF + MINT", *Protext III, Proceedings of the Third International Conference on Text Processing Systems*, J. J. H. Miller ed., Dublin, 22-24 October 1986, Boole Press 1986.

[Ayers] R. M. Ayers, J. Y. Horning, B. W. Lampson, J. G. Mitchell, *Interscript : A Proposal for a Standard for the Interchange of Editable Documents*, Xerox Palo Alto Research Center, Palo Alto, California, 1984.

[Baudelaire] P. C. Baudelaire, "Draw", *Alto User's Handbook*, B. W. Lampson and E. A. Taft ed., Xerox Palo Alto Research Center, November 1978, pp. 97-128.

[Beach] R. Beach, M. Stone, "Graphical Style Towards High Quality Illustrations", *Computer Graphics*, vol. 17, num. 3, July 1983, pp. 127-135.

[Bentley] J. L. Bentley, B. W. Kernighan, "GRAP - A Language for Typesetting Graphs", *Communications of the ACM*, vol. 29, num. 8, August 1986, pp. 782-792.

[Bigelow 83a] C. Bigelow, "Les caractères rationalisés", *Actes des Journées sur la manipulation de documents*, J. André ed., INRIA, Rennes, Mai 1983, pp. 15-27.

[Bigelow 83b] C. Bigelow, D. Day, "La typographie numérique", *Pour la Science*, Octobre 1983, pp. 48-62.

[Bigelow 85] C. Bigelow, "Font Design for Personal Workstations", *Byte*, January 1985, pp. 255-270.

[Bigelow 86] C. Bigelow, K. Holmes, "The Design of Lucida : an Integrated Family of Types for Electronic Literacy", *Text Processing and Document Manipulation, Proceedings of the International Conference*, J. C. van Vliet ed., Cambridge University Press, 1986, pp. 1-17.

[Blomberg] L. Blomberg, K. Frenckner, B. Kruse, G. Lönnemark, S. Romberger, Y. Sundblad, "A New Approach to Text and Image Processing", *IEEE Computer Graphics and Applications*, vol. 4, num 7, July 1984, pp. 12-22.

[Bogo] G. Bogo, H. Richy, I. Vatton, "Un modèle de représentation de documents généralisés", *Actes des journées sur la manipulation de documents*, J. André ed., INRIA, Rennes, Mai 1983, pp. 221-235.

[Bono] P. R. Bono, "A Survey of Graphics Standards and Their Role in Information Interchange", *Computer*, vol. 18, num. 10, October 1985, pp. 63-75.

[Buckle] N. Buckle, "Word Hyphenation in French", *Protext I, Proceedings of the First International Conference on Text Processing Systems*, J. J. H. Miller ed., Dublin, 24-26 October 1984, Boole Press 1984, pp. 108-113.

[CCITT] C.C.I.T.T., *Recommendation T.73, Document Interchange Protocol for the Telematic Services''*, C.C.I.T.T., TC29/1984/25D, Geneva, March 1984.

[Chamberlin] D. D. Chamberlin, J. C. King, D. R. Slutz, S. J. P. Todd, B. W. Wade, "JANUS: An Interactive System for Document Composition", *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, Portland, June 1981, et *ACM SIGPLAN Notices*, vol. 16, num. 6, 1981, pp. 82-91.

[Cherry] L. Cherry, "Computer Aids for Writers", *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, Portland, June 1981, et *ACM SIGPLAN Notices*, vol. 16, num. 6, 1981, pp. 61-67.

[Coray] G. Coray, R. Ingold, C. Vanoirbeek, "Formatting Structured Documents: Batch versus Interactive?" *Text Processing and Document Manipulation, Proceedings of the International Conference*, J. C. van Vliet ed., Cambridge University Press, 1986, pp. 154-170.

[Coutaz] J. Coutaz, "Abstractions for User Interface Design", *Computer*, vol. 18, num. 9, September 1985, pp. 21-34.

[Cowan] D. D. Cowan, G. DE V. Smit, "Combining Interactive Document Editing with Batch Document Formatting", *Text Processing and Document Manipulation, Proceedings of the International Conference*, J. C. van Vliet ed., Cambridge University Press, 1986, pp. 140-153.

[Crisanti] E. Crisanti, A. Formigoni, P. La Bruna, "EasyTEX: Towards interactive formulae input for scientific documents input with TEX", *TEX for Scientific Documentation*, Lecture Notes in Computer Science, vol. 236, edited by J. Désarménien, Springer-Verlag, 1986, pp. 55-64.

[Désarménien] J. Désarménien, "La division par ordinateur des mots français : application à TEX", *T.S.I.* vol. 5, num. 4, 1986, pp. 251-265.

[Dewey] M. E. Dewey, "TBLP - A Simple Preprocessor for Tables", *Software - Practice and Experience*, vol. 16, num. 8, August 1986, pp. 731-738.

[Donzeau] V. Donzeau-Gouge, G. Kahn, B. Lang, B. Mélèse, E. Morcos, "Outline of a tool for document manipulation", *IFIP 83*, Paris, R.E.A. Mason ed., Elsevier Science Publishers B.V., 1983, pp. 615-620.

[Dreyfus] M. Dreyfus, "Mathor", *Micro-Bulletin*, Num. 19, Avril 1986, C.N.R.S., pp. 23-36.

[ECMA] E.C.M.A., *Standard ECMA-101 Office Document Architecture*, E.C.M.A., Genève, Septembre 1985.

[Estublier] J. Estublier, S. Krakowiak, J. Mossière, Y. Rouzaud, "Design Principles of the Adèle Programming Environment", *International Computing Symposium on Application Systems Development*, ACM, March 1983.

[Flowers] J. Flowers, "Digital Type Manufacture: An Interactive Approach", *IEEE Computer*, May 1984, pp. 40-48.

[Foata] D. Foata, J.-J. Pansiot, Y. Roy, *Stratèc, un logiciel de traitement de textes mathématiques en amont de TEX*, Publ. I.R.M.A. Strasbourg 255/D-06, Décembre 1984.

[Foley] J. D. Foley, A. Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, Massachusetts, 1982.

[Furuta 82] R. Furuta, J. Scofield, A. Shaw, "Document Formatting Systems: Survey, Concepts, and Issues", *ACM Computing Surveys*, vol. 14, num. 3, September 1982, pp. 417-472.

[Furuta 86a] R. Furuta, "An Integrated, but not Exact-Representation, Editor/Formatter", *Text Processing and Document Manipulation, Proceedings of the International Conference*, J. C. van Vliet ed., Cambridge University Press, 1986, pp. 246-259.

[Furuta 86b] R. K. Furuta, *An Integrated, but not Exact-Representation, Editor/Formatter*, University of Washington, Technical Report 86-09-08, September 1986.

[Ganzinger] H. Ganzinger, W. Wilmertinger, "FOAM: A Two-Level Approach to Text Formatting on a Microcomputer System", *Software - Practice and Experience*, vol. 15, num. 4, April 1985, pp. 327-342.

[Gates] Y. Gates, "L'édition électronique", *La Recherche*, vol. 16, num. 163, Février 1985, pp. 262-271.

[GKS] International Standard Organisation, *Graphical Kernel System (GKS)*, Version 7.2, ISO, 1982.

[Goldberg] A. Goldberg, D. Robson, *Smalltalk-80: The Language and Its Implementation*, Addison-Wesley, Reading, Massachusetts, 1983.

[Goldfarb] C. F. Goldfarb, "A Generalized Approach to Document Markup", *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, Portland, June 1981, et *ACM SIGPLAN Notices*, vol. 16, num. 6, 1981, pp. 68-73.

[Gosling 81] J. Gosling, "A Redisplay Algorithm", *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, Portland, June 1981, et *ACM SIGPLAN Notices*, vol. 16, num. 6, 1981, pp. 123-129.

[Gosling 84] J. Gosling, "An Editor Based User Interface Toolkit", *Protext I, Proceedings of the First International Conference on Text Processing Systems*, J. J. H. Miller ed., Dublin, 24-26 October 1984, Boole Press 1984, pp. 126-132.

[Grevisse] M. Grevisse, *Le bon usage*, onzième édition revue, Duculot, Bruxelles, 1980.

[Gutknecht 84] J. Gutknecht, W. Winiger, "Andra: The Document Preparation System of the Personal Workstation Lilith", *Software - Practice and Experience*, vol. 14, 1984, pp. 73-100.

[Gutknecht 85] J. Gutknecht, "Concepts of the Text Editor Lara", *Communications of the ACM*, vol. 28, num. 9, September 1985, pp. 942-960.

[Habermann] A. N. Habermann, D. Notkin, "Gandalf: Software Development Environments", *IEEE Transactions on Software Engineering*, vol. 12, num. 12, December 1986, pp. 1117-1127.

[Hamlet] R. Hamlet, "A Disciplined Text Environment", *Text Processing and Document Manipulation, Proceedings of the International Conference*, J. C. van Vliet ed., Cambridge University Press, 1986, pp. 78-89.

[Hammer] M. Hammer, R. Ilson, T. Anderson, E. Gilbert, M. Good, B. Niamir, L. Rosenstein, S. Schoichet, "The Implementation of Etude, An Integrated and Interactive Document Production System", *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, Portland, June 1981, et *ACM SIGPLAN Notices*, vol. 16, num. 6, 1981, pp. 137-146.

[Harslem] E. Harslem, L. E. Nelson, "A Retrospective on the Development of Star", *Proceedings of the 6th International Conference on Software Engineering*, Tokyo, September 1982, pp. 377-383.

[Hégron] G. Hégron, "Algorithmes de génération de caractères", *Typographie et Informatique*, Support du cours INRIA, Rennes 21-25 Janvier 1985, pp. 53-76.

[Hershey] W. R. Hershey, "Thinktank, an outlining and organizing tool", *Byte*, May 1984, pp. 189-194.

[Hibbard] P. Hibbard, *User Manual for Mint - The Spice Document Preparation System*. Spice Document S153, Carnegie-Mellon University, April 1983.

[Horak] W. Horak, "Office Document Architecture and Office Document Interchange Formats: Current Status of International Standardization", *IEEE Computer*, vol. 18, num. 10, October 1985, pp. 50-62.

[Icon] Icon Technology Ltd, *MacAuteur, Manuel d'utilisation*, Italsoft, Paris, 1986.

[ISO 85] I.S.O., *Information Processing - Text and office systems - Standard Generalized Markup Language (SGML)*, Draft international standard ISO/DIS 8879, 1985.

[ISO 86] I.S.O., *Information Processing - Text and Office Systems - Office Document Architecture (ODA) and Interchange Format*. ISO/DIS 8613, 1986.

[Joloboff 83] V. Joloboff, "An Interactive Graphics Editor for Document Preparation", *Proceedings of 1983 ACM Conference on Personal and Small Computers*, San Diego, December 1983, and *ACM SigPC Notices*, vol. 6, num. 6, pp. 45-53.

[Joloboff 84] V. Joloboff, V. Quint, "Two-dimensional editing", *Proceedings of the IEEE First International Conference on Office Automation*, New Orleans, December 1984, pp. 41-49.

[Joloboff 86] V. Joloboff, "Trends and Standards in Document Representation", *Text Processing and Document Manipulation, Proceedings of the International Conference*, J. C. van Vliet ed., Cambridge University Press, 1986, pp. 107-124.

[Joloboff 87] V. Joloboff, "Représentation des documents : Etat de l'art, Recherche", *Cours INRIA Structure de/for Documents*. INRIA, Janvier 1987, pp. 81-103.

[Karov] P. Karov, *IKARUS for typefaces in digital form*, URW Ubternehmensberatung, Hamburg, 1983.

[Kasik] D. J. Kasik, "A User Interface Management System", *Siggraph'82 Conference Proceedings*, R. Daniel Bergeron ed., *Computer Graphics*, vol. 16, num. 3, July 1982, pp. 99-106.

[Kernighan 75] B. W. Kernighan, L. L. Cherry, "A System For Typesetting Mathematics", *Communications of the ACM* vol. 18, num. 3, March 1975, pp. 151-157.

[Kernighan 78] B. W. Kernighan, M. E. Lesk, J. F. Ossana, "UNIX Time-Sharing System: Document Preparation", *The Bell System Technical Journal*, vol. 57, num. 6, 1978, pp. 2115-2135.

[Kernighan 82] B. W. Kernighan, "PIC - A Language for Typesetting Graphics", *Software - Practice and Experience*, vol. 12, 1982, pp. 1-21.

[Kimura] G. D. Kimura, *A Structure Editor and Model for Abstract Document Objects*. University of Washington, Department of Computer Science, Technical Report 84-07-04, July 1984.

[King 84] P. R. King, "W-p: A prototype text-processor and document formatter", *Protext I, Proceedings of the First International Conference on Text Processing Systems*, J. J. H. Miller ed., Boole Press, Dublin, 1984, pp. 161-172.

[King 86] P. R. King, "An Overview of the W Document Preparation System", *Text Processing and Document Manipulation, Proceedings of the International Conference*, J. C. van Vliet ed., Cambridge University Press, 1986, pp. 188-199.

[Knuth 84] D. E. Knuth, *The TEXbook*, Addison-Wesley, Reading, Massachusetts, 1984.

[Knuth 85] D. E. Knuth, *The Metafont book*, Addison-Wesley, Reading, Massachusetts, 1985.

[Lamport] L. Lamport, *LaTEX: A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, 1986.

[Lampson] B. W. Lampson, "Bravo Manual", *Alto User's Handbook*, B. W. Lampson and E. A. Taft ed., Xerox Palo Alto Research Center, November 1978.

[Lesk] M. E. Lesk, "TBL A Program to Format Tables", *UNIX Programmer's Manual*, 7th edition volume 2A, Bell Telephone Laboratories, January 1979.

[Levison] M. Levison, "Editing Mathematical Formulae", *Software - Practice and Experience*, vol. 13, 1983, pp. 189-195.

[Liang] F. M. Liang, *Word Hy-phen-a-tion by Com-put-er*. Stanford University, Report STAN-CS-83-977, August 1983.

[Lopez] M. Lopez, J. Palazzo Oliveira, F. Velez, *The TIGRE Data Model*, Rapport de Recherche TIGRE n^{o} 2, IMAG, Grenoble, Novembre 1983.

[Marchand] J. Marchand, E. Bessone, *INTERMATH*. Rapport de projet de 3ème année, ENSIMAG, Grenoble, Juin 1985.

[Medina] R. Medina-Mora, *Syntax directed editing: toward integrated programming environments*, CMU-CS-82-113, Carnegie-Mellon University, 1982.

[Mélèse] B. Mélèse, "Edition de Documents Multi-langages sous Mentor-Rapport", *T.S.I.* vol. 4, num. 5, 1986, pp. 267-277.

[Meyrowitz] N. Meyrowitz, A. van Dam, "Interactive Editing Systems", *ACM Computing Surveys*, vol. 14, num. 3, September 1982, pp. 321-415.

[Morris 85] R. Morris, "Is what you see enough to get ? A description of the Interleaf Publishing System", *Protext II. Proceedings of the Second International Conference on Text Processing Systems*, J. J. H. Miller ed., Dublin, 23-25 October 1985, Boole Press 1985.

[Morris 86] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. H. Rosenthal, F. Donelson Smith, "Andrew: A Distributed Personal Computing Environment", *Communications of the ACM*, vol. 29, num. 3, March 1986, pp. 184-201.

[Nanard 84] J. Nanard, M. Nanard, "Manipulation interactive de documents", *T.S.I*, vol. 3, n. 6, 1984, pp. 443-451.

[Nanard 86] J. Nanard, M. Nanard, G. Cottin, "Pleiade, a system for interactive manipulation of structured documents", *Conference on Workstations and Publication Systems*, British Computer Society, London, October 22-23 1986.

[Nedginn] P. J. Nedginn, T. L. Bworn, "Clog: an Ada Package for Automatic Footnote Generation in Unix", *Communications of the ACM*, vol. 27, num. 4, April 1984, p. 351.

[Nelson] G. Nelson, "Juno, a constraint-based graphics system", *Computer Graphics*, vol. 19, num. 3, Siggraph'85 Conference Proceedings, pp. 235-243, 1985.

[Palais] R. S. Palais, "Mathematical Text Processing", *Notices of the AMS*, January 1986, pp. 3-37.

[Paris] J. Paris, L. Delamare, "L'environnement document sur le poste de travail Concerto", *Actes des journées Concerto*, Perros-Guirec, 4-6 Février 1986, CNET, pp. 191-217.

[Peels] A. Peels, N. Janssen, W. Nawijn, "Document Architecture and Text Formatting", *ACM Transactions on Office Information Systems*, vol. 3, n. 4, 1985, pp. 347-369.

[Pike] R. Pike, "Graphics in Overlapping Bitmap Layers", *ACM Transactions on Graphics*, vol. 2, num. 2, April 1983, pp. 135-160.

[Plass] M. E. Plass, D. E. Knuth, "Choosing Better Line Breaks", *Document Preparation Systems*, edited by J. Nievergelt, G. Coray, J.-D. Nicoud, A.C. Shaw, North-Holland, 1982, pp. 221-242.

[Quint 82] V. Quint, "Editing Mathematics on the Buroviseur", *Office Information Systems*, N. Naffah ed., North-Holland, 1982, pp. 149-159.

[Quint 83] V. Quint, "Un système interactif pour la production des documents mathématiques", *T.S.I.* vol. 2, num. 3, 1983, pp. 179-190.

[Quint 84] V. Quint, "Interactive Editing of Mathematics", *Protext I, Proceedings of the First International Conference on Text Processing Systems*, J. J. H. Miller ed., Boole Press, Dublin, 1984, pp. 55-68.

[Quint 86a] V. Quint, I. Vatton, "Grif: An Interactive System for Structured Document Manipulation", *Text Processing and Document Manipulation, Proceedings of the International Conference*, J. C. van Vliet ed., Cambridge University Press, 1986, pp. 200-213.

[Quint 86b] V. Quint, I. Vatton, H. Bedor, "Grif: an Interactive Environment for TEX", *TEX for Scientific Documentation*, Lecture Notes in Computer Science, vol. 236, edited by J. Désarménien, Springer-Verlag, 1986, pp. 145-158.

[Quint 86c] V. Quint, I. Vatton, H. Bedor, "Le système Grif", *T.S.I.* vol. 5, num. 4, 1986, pp. 337-341.

[Quint 86d] V. Quint, X. Rousset de Pina, *EDIMATH, Manuel d'utilisation*, Microsphère, Lyon, Juillet 1986.

[Quint 87a] V. Quint "Les systèmes pour la manipulation de documents structurés" *Cours INRIA Structure delfor Documents*. INRIA, Janvier 1987, pp. 39-80.

[Quint 87b] V. Quint, I. Vatton, "An Abstract Model for Interactive Pictures", soumis à *Interact'87*.

[Reid 80] B. K. Reid, "A high-level approach to computer document production", *Proceedings of the 7th Annual ACM Symposium on Principles of Programming Languages*, ACM SIGPLAN-SIGACT, January 1980.

[Reid 83] B. Reid, "Scribe : Histoire et évaluation", *Actes des Journées sur la manipulation de documents*, J. André ed., INRIA, Rennes, Mai 1983, pp. 28-39.

[Reid 86] B. K. Reid, "Procedural Page Description Languages", *Text Processing and Document Manipulation, Proceedings of the International Conference*, J. C. van Vliet ed., Cambridge University Press, 1986, pp. 214-223.

[Reiss] S. P. Reiss, "Graphical Program Development with PECAN Program Development Systems", *Proceedings of the ACM Sigsoft/Sigplan Software Engineering Symposium on Practical Software Development Environments*, P. Henderson ed., Pittsburgh, April 23-25, 1984. Egalement dans *ACM Sigplan Notices*, vol. 19, num. 5, May 1984, et dans *ACM Software Engineering Notes*, vol. 9, num. 3, May 1984.

[Ritchie] D. M. Ritchie, "UNIX Time Sharing System: A retrospective." *The Bell System Technical Journal*, vol. 57, num. 6, July-August 1978, pp. 1947-1969.

[Seybold] J. W. Seybold, "Document Preparation Systems and Commercial Typesetting", *Document Preparation Systems*, edited by J. Nievergelt, G. Coray, J.-D. Nicoud, A.C. Shaw, North-Holland, 1982, pp. 243-264.

[Smith] D. C. Smith et al., "Designing the Star User Interface", *Byte*, February 1982, pp. 242-282.

[Sproull] R. F. Sproull, B. K. Reid, *Introduction to Interpress*, XSIG 038404, Xerox Corporation, El Segundo, California, 1984.

[Stallman] R. M. Stallman, "EMACS, The Extensible, Customizable, Self-Documenting Display Editor", *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, Portland, June 1981, et *ACM SIGPLAN Notices*, vol. 16, num. 6, 1981, pp. 147-156.

[Sun] Sun Microsystems, Inc., *NeWS Preliminary Technical Overview*, Sun Microsystems, Inc., Mountain View, CA, October 1986.

[Sundblad] Y. Sundblad, "Computer Based Tools for Skilled Page Make-up Work", *Protext I, Proceedings of the First International Conference on Text Processing Systems*, J. J. H. Miller ed., Boole Press, Dublin, 1984, pp. 227-233.

[Tazi] S. Tazi, J. Virbel, "Formal Representation of Textual Structures for an Intelligent Text-Editing System", *Proceedings of the Workshop on Natural Language Understanding and Logic Programming*, North-Holland, 1985.

[Teitelbaum] T. Teitelbaum, T. Reps, "The Cornell program synthesizer: a syntax directed programming environment", *Communications of the ACM*, vol. 24, num. 9, September 1981, pp. 563-573.

[Thacker] C. P. Thacker, E. M. McCreight, B. W. Lampson, R. F. Sproull, D. R. Boggs, *Alto: A personal computer*, Technical Report CSL-79-11, Xerox Palo Alto Research Center, August 1979.

[Traband] P. Traband, "Fonctions typographiques en photocomposition programmée", *Actes des Journées sur la manipulation de documents*, J. André ed., INRIA, Rennes Mai 1983, pp. 3-14.

[Velez] F. Velez, *Un modèle et un langage pour les bases de données généralisées. Projet Tigre*. Thèse de Docteur-Ingénieur, Institut National Polytechnique de Grenoble, Septembre 1984.

[Verges] J.-C. Verges-Escuin, J.-P. Verjus, "Reconnaissance automatique des structures des textes en vue de l'édition", *RAIRO*, Octobre 1973, pp. 85-120.

[van Vliet] J. C. van Vliet, J. B. Warmer, "An Annoted Bibliography on Document Processing", *Text Processing and Document Manipulation, Proceedings of the International Conference*, J. C. van Vliet ed., Cambridge University Press, 1986, pp. 260-276.

[Walker] J. H. Walker, "The Document Editor: A Support Environment for Preparing Technical Documents", *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, Portland, June 1981, et *ACM SIGPLAN Notices*, vol. 16, num. 6, 1981, pp. 44-50.

[Warnock 80] J. E. Warnock, "The Display of Characters Using Gray Level Sample Arrays", *Computer Graphics*, vol. 14, num. 3, July 1980, pp. 302-307.

[Warnock 82] J. E. Warnock, D. K. Wyatt "A Device Independent Graphics Imaging Model for Use with Raster Devices", *Computer Graphics*, vol. 16, num. 3, July 1982, pp. 313-319.

[Williams] G. Williams, "The Apple Macintosh Computer", *Byte*, February 1984, pp. 30-54.

[Wong] P. C. S. Wong, E. R. Reid, "Flair - User Interface Dialog Design Tool", *Siggraph'82 Conference Proceedings*, R. Daniel Bergeron ed., *Computer Graphics*, vol. 16, num. 3, July 1982, pp. 87-98.

[Woodman] M. Woodman "Formatting Syntactically Nested Documents", *Protext I, Proceedings of the First International Conference on Text Processing Systems*, J. J. H. Miller ed., Boole Press, Dublin, 1984, pp. 240-246.

[van Wyk] C. J. van Wyk, "A Graphics Typesetting Language", *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, Portland, June 1981, et *ACM SIGPLAN Notices*, vol. 16, num. 6, 1981, pp. 99-107.

[Xerox] Xerox Corporation, *Interpress Electronic Printing Standard*, Document number XSIS 04804, Stamford, Connecticut, 1984.

TABLE DES MATIERES

INTRODUCTION **1**

1. Le sujet 1
2. Plan de la thèse 2

CHAPITRE 1 TYPOGRAPHIE ET INFORMATIQUE **5**

1. UN HISTORIQUE 5

1.1. L'évolution du matériel de typographie 5
1.1.1. La composition chaude 6
1.1.2. La première génération de photocomposeuses 7
1.1.3. La deuxième génération 7
1.1.4. La troisième génération 8
1.1.5. La quatrième génération 8
1.2. L'évolution des outils informatiques. 9
1.2.1. Les imprimantes et les écrans 9
1.2.2. L'évolution des logiciels de production de documents 11

2. LES OUTILS INFORMATIQUES DE LA TYPOGRAPHIE 12

2.1. Le traitement des caractères 13
2.2. La composition 16
2.3. La mise en page 18

3. LES SYSTEMES DE PRODUCTION DE DOCUMENTS 19

3.1. Une taxinomie des systèmes de production de documents 19
3.1.1. Les éditeurs et systèmes de traitement de texte 19
3.1.2. Les formateurs 21
3.1.3. Les éditeurs syntaxiques 22
3.1.4. Les éditeurs et formateurs spécialisés 23
3.1.5. Les interfaces utilisateur 25
3.2. Scribe 25
3.2.1. Mint 28
3.2.2. LaTeX 30
3.3. Interleaf 30

3.3.1. Le gestionnaire de documents 31
3.3.2. Le modèle de document 31
3.3.3. Le formatage des documents 32
3.3.4. L'édition du document 33
3.3.5. L'éditeur graphique 35
3.3.6. Conclusion 36
3.4. Les normes régissant les documents 36
3.4.1. Les normes pour les documents révisables. 37
3.4.2. Les normes pour les documents formatés. 41

4. CONCLUSION 43

CHAPITRE 2 UNE PREMIERE APPROCHE : LE TRAITEMENT DES FORMULES 45

1. PRESENTATION 45

1.1. Les outils existants 45
1.2. Une autre approche 48

2. LA STRUCTURE DES FORMULES 50

2.1. Les principes de structuration 50
2.2. Les constructions d'Edimath 52

3. LES DIFFERENTES REPRESENTATIONS DES FORMULES 58

3.1. La représentation interne 58
3.2. La représentation graphique 61
3.3. La représentation externe 65

4. L'INTERFACE UTILISATEUR D'EDIMATH 69

4.1. La saisie d'une formule 69
4.2. Les modifications de formules 72

5. LA MISE EN OEUVRE D'EDIMATH 74

5.1. L'architecture 74
5.2. Les principes de fonctionnement 75
5.3. Le réaffichage 78

6. LES DEVELOPPEMENTS 83

6.1. Les portages 83
6.2. La connexion à d'autres applications 85

6.3. La connexion à TEX 86

7. CONCLUSION 86

7.1. Les qualités et les défauts d'Edimath 86
7.2. Pour un système général 88

CHAPITRE 3 UNE APPROCHE GLOBALE : DOCUMENTS ET OBJETS STRUCTURES 91

1. PRESENTATION 91

2. UN MODELE SIMPLE 91

3. UN MODELE DE HAUT NIVEAU 94

3.1. Un modèle logique 94
3.2. Un modèle ouvert 95
3.3. Un modèle séparant les structures logique et physique 97
3.4. Un modèle général 99

4. LA META-STRUCTURE DES DOCUMENTS 100

4.1. Les types de base 101
4.2. Les éléments construits 102
4.3. Les constructeurs 103
4.4. Les éléments associés 105
4.5. Les attributs 106
4.6. Discussion du modèle 107

5. UN LANGAGE POUR DEFINIR LES STRUCTURES GENERIQUES 109

5.1. L'organisation générale d'un schéma de structure 110
5.2. La présentation par défaut 110
5.3. Les attributs 111
5.4. Les paramètres 111
5.5. Les éléments structurés 111
5.6. Les définitions de structure 112
5.7. Les importations 114
5.8. Les éléments associés 114
5.9. Les unités exportées 115

6. QUELQUES EXEMPLES 115

6.1. Une classe de documents : les articles 115
6.2. Une classe d'objets : les formules d'Edimath 118

7. LA PRESENTATION DES DOCUMENTS 119

7.1. Une présentation à deux niveaux 120
7.2. Les boîtes 121
7.3. Les vues et la visibilité 123
7.4. La numérotation 124
7.5. Les paramètres de présentation 125

8. UN LANGAGE POUR DECRIRE LA PRESENTATION 126

8.1. L'organisation d'un schéma de présentation 126
8.2. Les vues 127
8.3. Les compteurs 128
8.4. Les constantes de présentation 130
8.5. Les variables 131
8.6. Les règles de présentation par défaut 133
8.7. Les boîtes de présentation et de mise en page 133
8.8. La présentation des éléments structurés 134
8.9. La présentation des attributs 134
8.10. Les règles de présentation 135
8.11. Une règle de présentation 136
8.12. Les axes 137
8.13. Les positionnements relatifs 139
8.14. Les dimensions des boîtes 141
8.15. L'héritage 143
8.16. Les valeurs numériques 144
8.17. Les Booléens 146
8.18. Le retrait 146
8.19. Le contenu 147
8.20. La création de boîtes de présentation 148
8.21. La mise en page 151
8.22. Les copies de boîtes 153

9. CONCLUSION 154

CHAPITRE 4 L'EDITEUR DE DOCUMENTS GRIF 155

1. PRESENTATION 155

2. LES PRINCIPES D'EDITION 155

2.1. L'image du document 156
2.2. Les commandes et les traitements 157

2.3. L'impression des documents 158

3. LES FONCTIONALITES DE L'EDITEUR 160

3.1. Les vues 160
3.2. L'édition 161

4. L'ARCHITECTURE GENERALE DE GRIF 162

5. LES COMPILATEURS 164

5.1. Les principes de construction des compilateurs 164
5.2. Le compilateur de schémas de structure 166
5.3. Le compilateur de schémas de présentation 168

6. L'INTERFACE EDITEUR-MEDIATEUR 169

6.1. Les images abstraites 170
6.2. Les fenêtres 177
6.3. Les messages 177
6.4. L'affichage et le réaffichage 178
6.5. La sélection 179
6.6. Les commandes 181
6.6.1. Les touches de fonction 181
6.6.2. Les catalogues 182
6.7. La saisie du contenu 184
6.8. Les autres appels du Médiateur 185
6.9. Quelques remarques sur l'interface Médiateur-Editeur 186

7. L'EDITEUR 187

7.1. Les structures de données 187
7.1.1. Les tables de structure et de présentation 189
7.1.2. Les images abstraites 189
7.1.3. Les arbres abstraits 190
7.2. Les manipulations de structure 196
7.2.1. La création des arbres abstraits 196
7.2.2. La sélection structurelle 197
7.2.3. Le parcours du document 199
7.2.4. Les modifications des arbres abstraits 201
7.3. L'affichage et le réaffichage 205
7.4. Les éléments hors structure 208
7.4.1. Les éléments associés 208
7.4.2. Les paramètres 209

7.5. La représentation pivot 211

8. QUELQUES EXEMPLES 212

8.1. Différents types de documents 212
8.2. Une session d'édition 220

9. DISCUSSION 223

9.1. L'intégration 223
9.2. L'interface utilisateur 224
9.3. La structure 225
9.4. Les lacunes 225

CHAPITRE 5 CONCLUSION 227

1. LES DOMAINES D'UTILISATION 227

1.1. Un générateur d'éditeurs 227
1.2. Une aide à la saisie 228
1.3. La production de documents normalisés 229
1.4. Les métiers du livre 229
1.5. La documentation technique 230

2. COMPARAISON DE GRIF AVEC D'AUTRES SYSTEMES 231

2.1. Les systèmes retenus 231
2.2. La structure logique 234
2.3. L'image des documents 236
2.4. Les objets inclus 237
2.5. Le mode d'interaction 237
2.6. Les autres caractéristiques 238

3. LES POTENTIALITES DE L'EDITION STRUCTUREE 239

ANNEXES..... 241

ANNEXE 1 LE META-LANGAGE M 241

ANNEXE 2 LE LANGAGE S 243

ANNEXE 3 LE LANGAGE P 245

ANNEXE 4 LA REPRESENTATION PIVOT 251

ANNEXE 5 DONNEES TECHNIQUES 259
1. EDIMATH 259
2. GRIF 260
2.1. Les compilateurs 260
2.2. L'Editeur 260
2.3. Le Médiateur 261
2.4. Le Traducteur 262

REFERENCES BIBLIOGRAPHIQUES 263

TABLE DES FIGURES

1.1 Un exemple, l'environnement Itemize de Scribe 27
1.2 Un document décrit dans le langage Mint 29

2.1 Une formule et sa représentation interne 59
2.2 Deux représentations internes équivalentes 60
2.3 Les boîtes d'une formule 65
2.4 L'arbre et la liste des boîtes 76

3.1 Deux structures spécifiques 96
3.2 Les côtés et axes des boîtes 122
3.3 Position et dimension des boîtes 143
3.4 Création de boîtes de présentation 151

4.1 L'architecture de Grif 163
4.2 Principe de fonctionnement des compilateurs 165
4.3 Exemple de positionnement de deux pavés 172
4.4 Les structures de données de l'éditeur Grif 188
4.5 Le chaînage des références 193
4.6 Une sélection 198
4.7 L'image d'un document bilingue 213
4.8 L'image d'un formulaire 215
4.9 L'image d'une lettre publi-postée 216
4.10 L'image "finale" d'un article 218
4.11 L'image "de travail" d'un article 219
4.12 L'édition d'un article 220-222

Colophon

Le présent ouvrage a été écrit, au moins en partie, pendant le déroulement de l'étude. Sa réalisation n'a donc pas pu bénéficier de tous les résultats obtenus, et notamment l'éditeur Grif n'a été utilisé que pour produire les figures qui illustrent la fin du chapitre 4.

L'ensemble du document a été mis en page par Sroff, un formateur développé initialement pour offrir à Grif un moyen de sortie sur la seule imprimante à laser disponible dans notre environnement au début du projet : la LN01 connectée au VAX du Laboratoire de Génie Informatique. Sroff admet un large sous-ensemble des commandes de Troff et peut traiter les textes en français : il intègre les règles de coupure des mots et les règles typographiques propres à la langue française. Il traite également les schémas ; la plupart des figures de cette thèse ont été obtenues par ce moyen, à l'exception des copies d'écran. Il permet enfin d'inclure des formules qui ont été créées par Edimath : les formules du chapitre 2 ont toutes été éditées avec Edimath puis intégrées par Sroff.

Sroff utilise plusieurs polices, de corps et de chasse variable. Celles-ci ont été obtenues à partir des polices Times fournies dans la bande de distribution du système UNIX 4.2 BSD. Il a fallu convertir ces polices au format de l'imprimante. Les polices ainsi obtenues, provenant de systèmes d'origine américaine, ne comportaient aucun des signes diacritiques utilisés en français. Pour les compléter, nous avons utilisé un Perq, qui est doté d'un bon éditeur de polices, mais qui travaille sur son propre format de représentation des caractères. D'autres outils de conversion ont donc été nécessaires. Ce même éditeur a été utilisé pour créer les quelques polices spéciales dont Sroff se sert pour imprimer les graphiques et les formules. Il a également été employé pour créer les polices dont Edimath a besoin pour afficher des formules mathématiques sur le terminal VT220 (encore des conversions de format). Enfin, il a servi a créer tous les caractères et symboles de Grif.

L'édition du "compuscrit" de cette thèse a été réalisée sur VAX avec Elle, un des nombreux éditeurs dérivés d'Emacs. Le terminal était un VT220 téléchargé avec une police spéciale comportant tous les caractères accentués du français.

AUTORISATION DE SOUTENANCE

DOCTORAT D'ETAT

Vu les dispositions de l'Article 5 de l'Arrêté du 16 avril 1974,

Vu les rapports de M..G..CORAY.....

M..G..KAHN.....

M..S..KRAKOWIAK.....

M..Vincent.QUINT..... est autorisé à présenter une thèse en vue de l'obtention du grade de **DOCTEUR D'ETAT ES SCIENCES.**

Fait à Grenoble, le 3 AVR. 1987

Le Président de l'U.S.T.M.G.

UNIVERSITÉ SCIENTIFIQUE ET MÉDICALE DE GRENOBLE

J.J PAYAN

Titre :

Une approche de l'édition structurée des documents

Résumé :

L'édition d'un document peut être vue comme la manipulation d'une structure abstraite qui représente l'organisation logique des composants du document. A partir de ce principe, on propose un méta-modèle qui permet la description des structures logiques de toutes sortes de documents et de différents types d'objets fréquents dans les documents : formules mathématiques, tableaux, schémas, etc... On associe aux structures logiques des règles de présentation qui déterminent l'aspect graphique de leurs composants. On montre l'intérêt de cette approche en présentant deux systèmes interactifs construits sur ce modèle : l'éditeur de formules mathématiques Edimath et l'éditeur de documents Grif. La présentation de ces systèmes s'appuie sur un état de l'art de la typographie informatique.

Mots-clés :

Système interactif, manipulation de documents, édition structurée, interface homme-machine, modèle de document.