CEPAGE: UN EDITEUR STRUCTUREL PLEINE PAGE

Bertrand Meyer [+], Jean-Marc Nerson [++]

[+] Computer Science Department, University of California
Santa Barbara, California 93106 (Etats-Unis)
Tél. (805) 961 43 85[1]

[++] CIMSA, 10 Avenue de l'Europe
78140 Vélizy (France)[1]

RESUME[2]

Nous décrivons Cépage, un éditeur de documents structurés conçu pour etre d'emploi agréable sur les terminaux actuels. Cépage se trouve au confluent des travaux sur les éditeurs syntaxiques, du développement des éditeurs pleine page, et des études sur les environnements logiciels avancés. C'est un éditeur universel, dans lequel la description du langage est un simple paramètre ; son interface externe est faite pour les enfants de l'ère vidéo. Cépage constitue un prototype de ce que pourrait etre un éditeur structurel utilisable dans un environnement industriel.

MOTS-CLES

Editeurs structurels, éditeurs syntaxiques, ergonomie des dialogues, communication homme-machine, environnements de programmation, formatage des programmes.

ABSTRACT

This paper describes Cépage, an editor for structured documents, designed for ease of use on modern terminals. Cépage was conceived as the common child of three influences: syntax editors; full-screen editors; and advanced software environments. Cépage is universal, the language description being a mere parameter to the system; its user interface is intended to be acceptable to the children of the video game era. We think Cépage is a prototype of what a structural editor should look like in order to succeed in production environments.

KEYWORDS

Structural editors, Syntax Editors, Human Interfaces, Man-Machine Communication, Programming Environments, Program Formating, Ergonomics.

[1] Travail effectué initialement à: Electricité de France, Direction des Etudes et Recherches, 1 avenue du Général de Gaulle 92141 Clamart (France)

[2] Nous présentons nos excuses au lecteur pour les anomalies de typographie (accents circonflexes) et de bibliographie (mentions en anglais), dues au fait que cet article a été préparé aux Etats-Unis sur un système de traitement de textes mal adapté à la langue française.

CEPAGE: UN EDITEUR STRUCTUREL PLEINE PAGE

Bertrand Meyer
Jean-Marc Nerson

1. LES OBJECTIFS

Cépage est un éditeur structurel (terme que nous préférons à celui de "syntaxique"), dans la conception duquel l'interface humaine a été étudiée avec un soin tout particulier. Il est entièrement paramétrable et peut s'appliquer à tout langage défini par une grammaire: langage de programmation, de spécification, mais aussi langage de description de documents structurés de toute nature (nous appellerons ci-après "documents" les objets que l'éditeur sert à construire).

Cépage s'inscrit dans toute une lignée d'éditeurs structurels dévéloppés au cours des dernières années [Allison1983, Donzeau-Gouge1981, Donzeau-Gouge, Habermann1982, Hansen1971, 1981, Teitelbaum1981, Wilander1980, Teitelbaum1981] . Les éditeurs structurels, par opposition aux éditeurs de textes classiques, permettent de manipuler des documents non comme de simples suites de lignes ou de caractères, mais comme des objets structurés, en leur appliquant des opérations définies relativement à leur structure. Parmi les principaux avantages de cette méthode, on peut citer:

- la garantie d'obtenir des documents syntaxiquement corrects;
- la possibilité d'effectuer des transformations éventuellement complexes mais garanties correctes, par exemple des transformations de programmes en vue de leur optimisation ou de leur transport;
- la possibilité de décharger l'utilisateur d'une partie des tâches de routine liées à la nécessité, dans un éditeur de textes classique, de fournir tous les détails de la syntaxe "concrète" des documents;
- la possibilité de traduction automatique d'un cadre syntaxique dans un autre (par exemple dans le cas de conversions entre langages de programmation);
- l'utilisation d'une structure de données normalisée (en général l'arbre syntaxique abstrait) qui peut servir de support à d'autres outils logiciels (cf. par exemple [Schroeder1983]), voire à des environnements de programmation complets ([Habermann1982]).

En dépit de ces qualités, les éditeurs structurels n'ont pas encore gagné droit de cité dans l'industrie. L'une des raisons principales de cette situation est, selon nous, liée à leur interface externe qui, dans la plupart des cas, est de type "ligne à ligne", c'est-à-dire que le dialogue avec l'utilisateur consiste en une suite d'échanges de commandes et de réponses. Or les environnements de programmation disponibles aujourd'hui offrent de plus en plus couramment des éditeurs de texte **pleine page**, tels SPF (sur IBM), Emacs (sur Multics et Vax-Unix) ou Vi (sur Vax-Unix), qui tirent parti des possibilités des terminaux actuels. Parmi les caractéristiques de ces systèmes, on peut citer [Meyer1983a] :

- L'utilisation de l'écran complet, de préférence à la ligne, comme unité de communication entre le système et l'utilisateur, donnant à celui-ci une vision notablement plus large sur le document en cours de construction, et lui permettant donc d'exercer un meilleur contrôle sur l'ensemble du processus d'édition;

- la possibilité, plus facile à fournir que dans un système ligne à ligne, de personnaliser le dialogue en conservant des informations relatives à chaque utilisateur;

- l'utilisation en parallèle, dans certains systèmes, de plusieurs **fenetres**, permettant à l'utilisateur de posséder à chaque instant plusieurs vues différentes sur le document manipulé;

- enfin, et plus généralement, l'application du principe de "manipulation directe" [Shneiderman1983] , selon lequel on maîtrise mieux un système lorsqu'il fournit à chaque instant une représentation claire et à jour de l'état courant des objets traités.

Le bénéfice de ces différentes propriétés est tel qu'il est à peu près impossible de convaincre un utilisateur d'un éditeur pleine page de revenir à un éditeur ligne à ligne, quelles qu'en soient par ailleurs les qualités. Ceci, selon notre expérience, vaut aussi pour les éditeurs structurels: s'ils sont de type ligne à ligne, ils ne pourront gagner les faveurs des utilisateurs habitués à des systèmes pleine page.

Les objectifs de Cépage découlent des réflexions précédentes. Il s'agissait de combiner les avantages des éditeurs structurels en matière de sureté et de puissance avec la commodité d'emploi des éditeurs de textes pleine page, en tirant le meilleur parti possible des terminaux modernes.

Le projet Cépage ne se voulait pas un projet de recherche, mais plutot un transfert de technologie, destiné à rendre industriellement utilisables des idées, celles de l'édition structurelle, qui ont fait l'objet de travaux importants de la part des chercheurs. En fait, nous avons du, à notre corps défendant, "inventer" un peu plus que nous ne l'avions envisagé initialement.

Les principales sources d'inspiration ont été, pour les éditeurs structurels, Gandalf et (dans une moindre mesure) Mentor et CPS; comme modèle d'interface homme-machine, Smalltalk nous a également influencés.

Selon tout critère objectif, le projet Cépage est un petit projet. La spécification et la conception sont l'oeuvre des deux auteurs de cet article, la réalisation presque exclusivement du second (Cépage inclut un petit éditeur de textes, écrit par N. Triquet). Les premières discussions remontent à la fin de 1982; le projet a véritablement pris corps au début de 1983, avec pour objectif (qui a été respecté) d'obtenir un prototype en état de fonctionnement le 20 décembre 1983. La programmation proprement dite n'a commencé qu'en septembre 1983. Le programme comprend environ 6000 lignes en Pascal; il utilise par ailleurs le progiciel Gescran pour la gestion de l'interface écran [Audin1980] , réalisé dans la meme équipe, et qui représente environ 4000 lignes de Fortran 77 (Gescran est un ensemble de sous-programmes permettant de décrire commodément les interactions "plein écran" en ne manipulant que des objets appartenant à quatre types abstraits, appelés *écran*, *fenetre*, *zone*, *terminal* et accessibles uniquement à travers les primitives du progiciel [Meyer1982] ; il s'appuie sur le progiciel d'entrée et sortie Ensorcelé [Brisson1982, Meyer1981]). Les conditions quelque peu particulières dans lesquelles ce projet a été réalisé expliquent sans doute que ces paramètres ne correspondent guère à ce que l'on pourrait déduire de l'étude des bons auteurs [Boehm1982] .

Il peut etre intéressant de noter que l'utilisation partielle de spécifications formelles, fondées sur le langage Z [Abrial1980] puis sur la méthode M [Meyer1984a] , a rendu quelques services.

2. L'UTILISATION DE CEPAGE

2.1. L'écran

L'écran affecté à une session de Cépage est divisé en un certain nombre de **fenetres** (figure 1). Chacune de ces fenetres remplit une fonction précise:

- la fenetre "document" contient une représentation de l'état actuel du document en cours de construction ou de modification; certains des éléments de cette représentation, affichés entre chevrons (par exemple *instruction*), correspondent à des éléments du document qui n'ont pas encore été affinés et sont dits **non-terminaux**;

- la fenetre "texte" est destinée à recevoir les textes non structurés qu'il peut etre nécessaire de fournir à certaines étapes d'une session (par exemple des identificateurs, des commentaires);

- la fenetre "menu" offre à chaque étape la liste des choix disponibles;

- la fenetre "type" donne le type syntaxique des éléments délimités (cf. ci-après);

- des fenetres "réserves" (non présentes sur la figure 1) donnent des informations sur des documents ou éléments de documents autres que le document en cours d'édition; ces fenetres sont utilisées pour changer de document pendant la session ainsi que pour les opérations de copie et de transfert.

- la fenetre "message" sert à afficher les diagnostics.

2.2. Le dialogue

A chaque étape de l'exécution d'une session de Cépage, le système propose à l'utilisateur de choisir entre un certain nombre de possibilités à l'aide d'un menu. Pour utiliser les fonctions de base de Cépage, les menus suffisent; un manuel d'utilisation n'est donc pas nécessaire pour peu que l'on ait compris les concepts principaux du système. Dans la version IBM actuelle, le choix entre les différents éléments du menu s'effectue grace aux touches de fonction du terminal. Sur des terminaux plus évolués, on peut imaginer d'utiliser une souris.

Chaque fois qu'il est nécessaire de désigner un élément du document (par exemple pour indiquer à quel terminal s'applique un affinage, comme sur la figure 1a), on utilise à cet effet le curseur, que l'on positionne sur l'élément en question. C'est la seule facon d'accéder au document (la notion de numéro de ligne, par exemple, est absente). L'utilisation d'un dispositif plus rapide tel que la souris serait particulièrement bienvenue ici.

Quelques fonctions plus avancées exigent l'emploi de commandes; ces commandes sont formées d'un mot unique, et leur existence découle uniquement du nombre limité de touches de fonctions disponibles (12). Cépage n'a donc pas de "langage de commande" au sens classique du terme; toutes les interactions avec le système se font par "pointer-toucher".

En particulier, l'utilisateur construisant avec Cépage un texte de programme, en Pascal par exemple, n'est **jamais** amené à frapper au clavier des éléments de syntaxe concrète, par exemple des mots-clés tels qu' **if, procedure, record,** etc. Au lieu de cela, un menu lui permet de choisir entre *conditionnelle*, *déclaration de procédure*, *déclaration de type enregistrement*, etc., et le système produit pour lui la syntaxe correcte (les taches de routine sont l'affaire des ordinateurs, non celles des humains).

Le seul cas où le clavier (hors touches de fonction) est nécessaire est celui où l'utilisateur doit fournir un texte que le système ne pourrait inventer seul, comme un identificateur ou un commentaire. La fenetre "texte" est utilisée à cet effet; le texte y est construit grace à un éditeur de textes (pleine page) inclus dans Cépage.

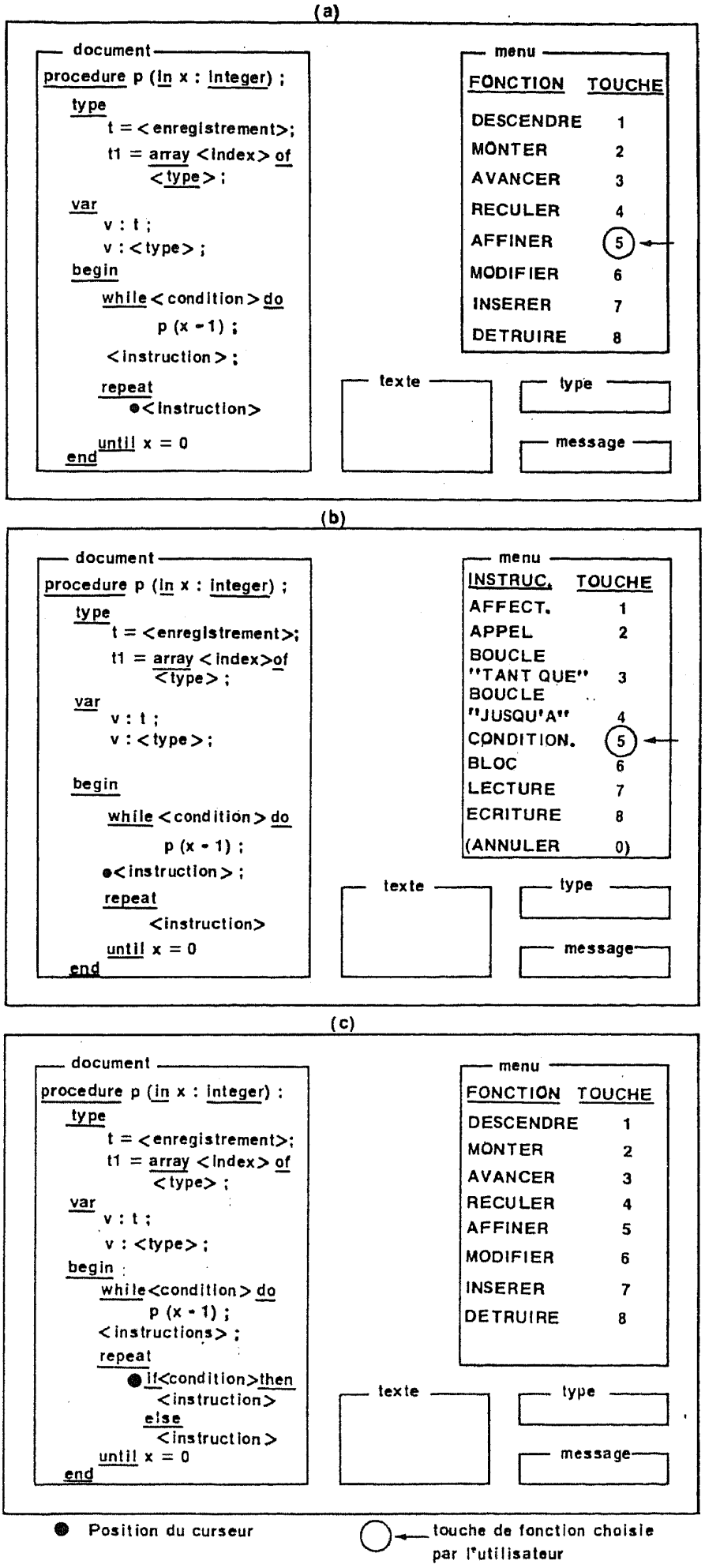


FIGURE 1 - UN AFFINAGE

2.3. Les fonctions de base

Les principales fonctions offertes par Cépage se rattachent aux catégories suivantes.

- **promenade:** parcours du document (montée et descente dans la hiérarchie des entités syntaxiques, avancée et récul dans les listes);

- **construction-modification:** affinage, changement d'un affinage antérieur, insertion et destruction dans une liste;

- **copie-transfert:** reproduction ou déplacement d'un élément de texte (utilisant l'opération de "délimitation": voir ci-après);

- **archivage-restauration:** archivage sur un fichier, sous une forme adéquate, de l'état actuel d'un document en cours d'élaboration, partiellement ou complètement affiné; restauration d'un document précédemment archivé.

- **génération:** production de la forme finale d'un document complètement affiné;

- **controle de session:** choix du document courant, passage d'un document à un autre, définition de bibliothèque etc. (une bibliothèque est un ensemble de documents; on peut au cours d'une session travailler sur plusieurs documents, dont un seul est actif à chaque instant, et passer librement de l'un à l'autre).

2.4. La délimitation

La délimitation (figure 2) est une opération nécessaire pour les fonctions qui exigent de l'utilisateur qu'il définisse un sous-ensemble syntaxique du document: ainsi, pour une copie ou un transfert, il faut délimiter la partie du document à laquelle s'appliquera l'opération. Cette délimitation s'effectue selon les principes de la manipulation directe.

Pour "délimiter", on place le curseur à un emplacement quelconque de l'élément à délimiter, et l'on précise la portée de ce document par une suite de commandes, effectuées grace aux touches de fonction (indiquées sur le menu de délimitation); à chaque étape, le système fait ressortir l'élément délimité par un changement des attributs d'affichage (couleur, affichage en négatif, etc.).

Les commandes de délimitation sont les suivantes:

- englober: inclure dans l'élément délimité la structure syntaxique immédiatement englobante (par exemple, si l'on avait jusque là délimité une instruction, inclure l'ensemble du bloc qui la contient);

- "désenglober": annuler l'effet d'une opération "englober" en revenant au niveau inférieur;

- étendre à gauche: inclure l'élément immédiatement antérieur (cette opération s'applique au cas où l'élément délimité est une sous-liste; les trois opérations complémentaires sont exclure à gauche, étendre à droite, exclure à droite);

- terminer (accepter l'élément actuel); annuler.

2.5. Modification du langage

Cépage est entièrement indépendant du langage; la syntaxe (concrète et abstraite) est un paramètre qui peut etre modifié à volonté. Dans la version actuelle, la description ou la modification du langage se fait de facon assez classique, par l'entrée d'une grammaire. Il est prévu ultérieurement de fournir pour cette opération l'interface du système lui-meme, ce qui revient à dire que l'un des langages pour lesquels Cépage sera défini est un langage de description syntaxique (il est bien conforme aux principes généraux de la conception de Cépage de faire en sorte que l'utilisateur n'ait pas à connaître la syntaxe concrète de ce "langage").

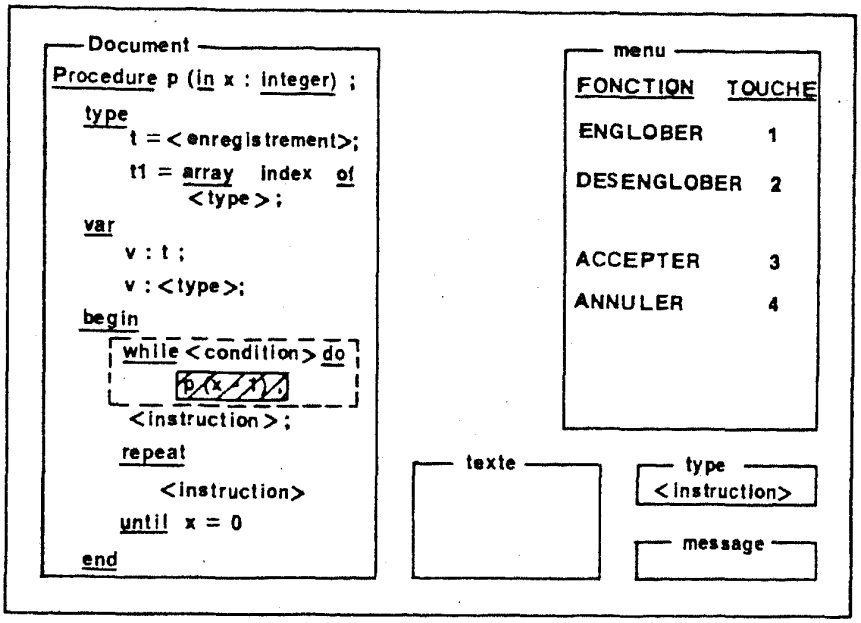


FIGURE 2 - LA DELIMITATION

La portion hachurée a été délimitée (et apparait en négatif ou dans une couleur spéciale). En appuyant sur la touche de fonction 1 ("ENGLOBER"), on délimite l'ensemble de la zone entourée en pointillé.

La modification du langage peut paraître une opération peu utile en pratique, pour autant que Cépage soit fourni avec des descriptions des principaux langages. En fait, la possibilité d'adapter facilement la description du langage à des conditions locales nous paraît une caractéristique vivement souhaitable. Elle permet en particulier de mettre en place des normes de programmation d'une facon plus commode (et plus facile à faire accepter) que par l'utilisation d'outils de controle *a posteriori*. On peut ainsi définir des sous-ensembles d'un langage, des conventions relatives aux commentaires, à la structure des programmes, etc.

3. CEPAGE: LES CHOIX TECHNIQUES

3.1. Les structures de données fondamentales

Au cours d'une session, Cépage travaille (figure 3) à partir de deux structures de données principales:

- la description interne du langage, ou graphe de grammaire;
- la description interne d'un ensemble de documents: foret syntaxique abstraite.

Il est important de noter que ces deux structures de données sont traitées sur un pied d'égalité. C'est ce qui fait de Cépage un système entièrement paramétré par le langage: la description du langage est interprétée répétitivement par le système. Ceci distingue nettement Cépage d'un système tel que Gandalf, paramétrable certes, mais

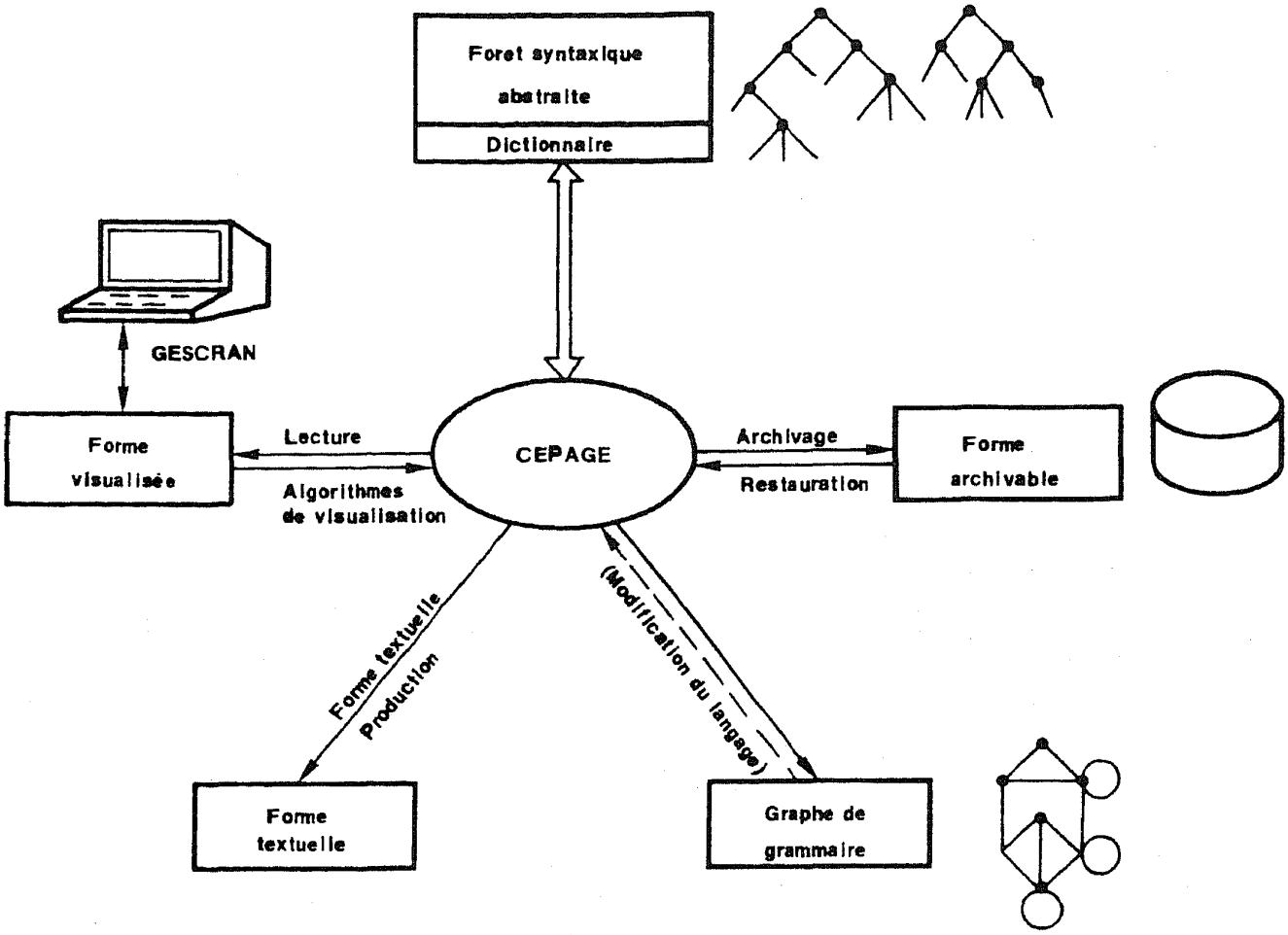


Figure 3 : Les structures de données

dans lequel la description du langage est "compilée", c'est-à-dire dans lequel on doit partir d'une version "noyau" de Gandalf et d'une description d'un langage X (ou Z, ou C) pour obtenir un outil Gandalf-X, ou Gandalf-C, adaptée au langage choisi. La solution adoptée par Cépage offre une plus grande souplesse et explique qu'il soit possible de modifier facilement le langage. En revanche, elle ne permet pas de prendre aussi facilement en compte des actions sémantiques, ce qui est un des buts de Gandalf.

Le **graphe de grammaire** est une structure de données permettant de représenter l'ensemble des propriétés de la grammaire du langage. La **syntaxe abstraite** est utilisée comme base; elle est décrite par un ensemble de **types syntaxiques** et de **productions**. Chaque type syntaxique apparaît à gauche d'une production au plus; ceux qui n'apparaissent à gauche d'aucune production sont dits terminaux. Il y a trois sortes de productions, dites "concaténation", "union" et "liste", illustrées respectivement par les exemples suivants:

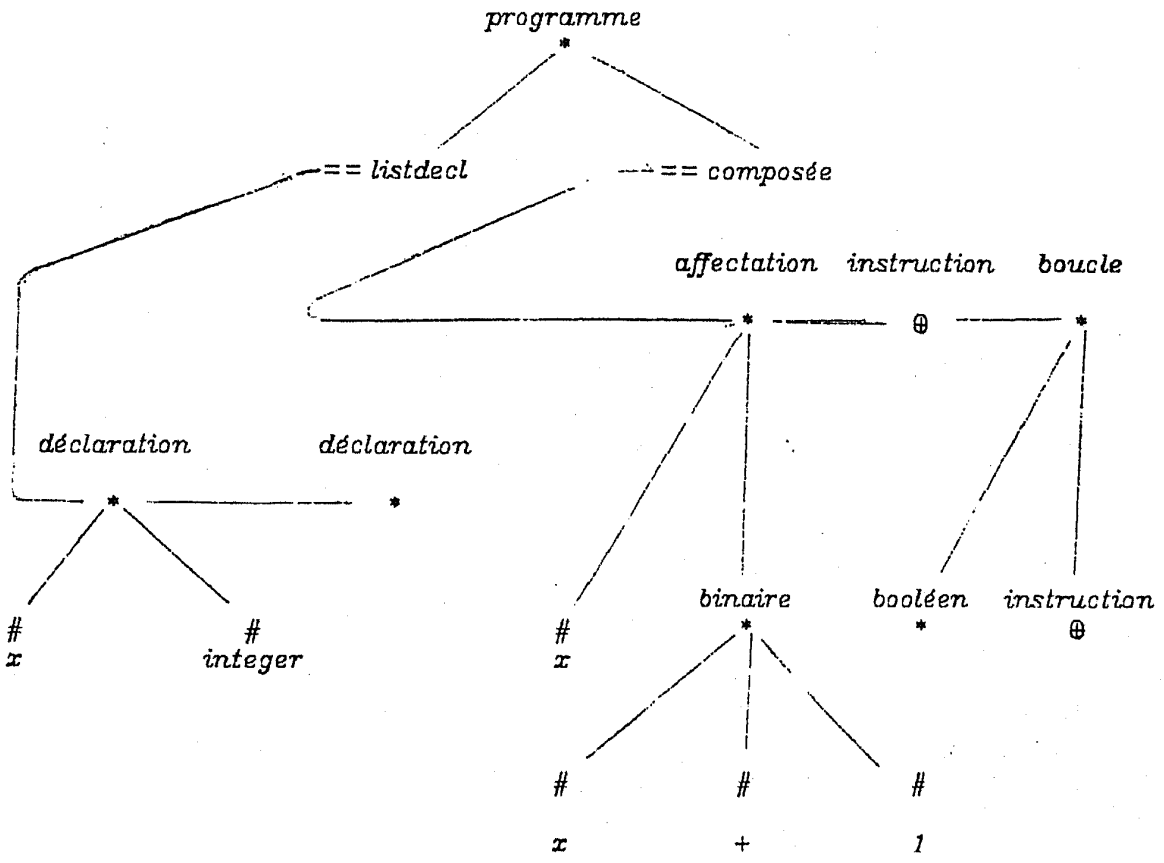
conditionnelle = c: *booléen* ; st1, st2: *instruction*;

instruction = *affectation* | *conditionnelle* | *composée*

composée = *instruction**

La **syntaxe concrète** est obtenue par "décoration" des productions de la syntaxe abstraite; par exemple, à toute production de type liste sont associés un en-tete, un délimiteur et une fin (par exemple **begin**, le point-virgule et **end** dans le cas de *composée*). Le graphe de grammaire regroupe l'ensemble de ces informations.

La **foret syntaxique abstraite** comprend un ensemble d'arbres syntaxiques abstraits, associés chacun à un document ou élément de document en cours d'élaboration.



* : noeud concaténé
== : noeud liste
⊕ : noeud alternatif
: noeud texte

Figure 4: Arbre Syntaxique Abstrait

Les noeuds internes d'un arbre syntaxique abstrait (figure 4) sont de quatre sortes, correspondant aux quatre types de productions:

- les noeuds "concaténés" ont une arité fixé;
- les noeuds "alternatifs" représentent seulement un choix dans une production de type union;
- les noeuds "liste" peuvent avoir un nombre quelconque de fils.

- les noeuds "texte", correspondant à des éléments terminaux affinés par l'utilisateur à l'aide de l'éditeur de textes inclus dans Cépage.

3.2. Autres structures de données

D'autres structures de données complètent les deux précédentes.

Outre les arbres syntaxiques abstraits, trois représentations sont nécessaires pour les documents:

- la forme visualisable, un ensemble d'éléments destinés à etre transmis à Gescran pour affichage sur le terminal à chaque étape de la session;

- la forme archivable, pour préservation et restauration ultérieure de l'état instantané d'un document;

- la forme textuelle, but ultime du processus d'édition.

Par ailleurs, la foret syntaxique abstraite s'accompagne d'un **dictionnaire**, contenant les différents éléments textuels nécessaires (identificateurs, etc.). Les feuilles des arbres syntaxiques contiennent des références au dictionnaire.

3.3. Les algorithmes

Il convient de faire remarquer que les objectifs définis précédemment impliquent **l'absence d'analyse syntaxique** dans Cépage. La construction d'un texte s'effectue par choix successifs, correspondant à la syntaxe abstraite; la syntaxe concrète est construite par le système, qui effectue en réalité l'opération inverse de l'analyse syntaxique, appelée parfois "désanalyse" (*un-parsing*).

On notera que la liberté laissée aux utilisateurs dans la description du langage permet d'établir en pratique un bon compromis entre la facilité d'utilisation et le degré de détail auquel descend le système; par exemple, on peut envisager de considérer *expression* comme un terminal. Une autre technique pour ce type d'entité syntaxique, non mise en oeuvre dans la version actuelle de Cépage, est celle de [Kaiser1982] , intermédiaire entre "analyse" et "déanalyse".

S'il n'y a pas d'analyse syntaxique, un autre type d'algorithmes a posé des problèmes sérieux: la construction de la forme visualisée. Il s'agit de proposer à chaque instant une représentation aussi riche que possible de l'état du document, en tenant compte des limites imposées par la taille physique du terminal.

Avec un éditeur de textes, pleine page ou non, on ne peut en général fournir qu'un extrait du document formé d'une suite contiguë de lignes (certains éditeurs offrent la possibilité d'exclure des groupes de lignes de la partie affichée afin de se concentrer sur les éléments les plus intéressants à un moment donné). Un éditeur structurel doit etre capable de fournir une vue globale du document ou d'une partie de celui-ci, meme s'il ne peut la représenter sur l'écran avec tous ses détails. La solution est l' **élision**: on remplace certains éléments du document par une abréviation - plus précisément, par une simple indication de leur type. Ainsi, une procédure de 2000 lignes pourra etre figurée par la simple indication *"procédure"*; nous appelons ce type d'abréviation **abstraction**. Le second type d'abréviation effectue par Cépage est le **rétrécissement**, qui consiste en une abstraction appliquée à une ou plusieurs sous-listes d'une liste, comme dans

"231 instructions";

p := *expression*;

"57 instructions"

A chaque étape de la session, le système détermine le **foyer** sur lequel l'utilisateur semble vouloir concentrer son attention d'après les dernières opérations qu'il a effectuées, et cherche à afficher une vue aussi détaillée que possible d'une portion du

document, de part et d'autre du foyer, déterminant les abstractions et retrecissements nécessaires. Il en déduit la forme visualisable qui est transmise à Gescran pour affichage.

La recherche d'une bonne représentation visualisable s'est révélée une tache d'une difficulté inattendue. Nous avons été surpris par le peu de documents disponibles; si l'on excepte une brève allusion dans [Barstow1984] , la seule référence publiée est à notre connaissance [Mikelsons1981] , qui est difficilement utilisable du fait de son imprécision et des caractéristiques particulières de l'environnement décrit.

L'abondante littérature sur le formatage des programmes ("prettyprinting", paragraphage) est ici de peu d'utilité; l'hypothèse fondamentale, quoique en général implicite (cf. en particulier [Oppen1980]) est que, si la longueur des lignes est limitée, le nombre de lignes, lui, ne l'est pas. Pour un formatage sur écran, **les colonnes et les lignes sont des ressources sévèrement limitées.**

Nous avons donc été amenés à concevoir des algorithmes spécialisés décrits ailleurs [Meyer1983b, Meyer1984b] , et qui dépassent le cadre de cet article. Ces algorithmes sont linéaires par rapport au nombre de noeuds de l'arbre syntaxique. Il s'agit de l'un des domaines où nous avons dû "inventer".

4. L'AVENIR DE CEPAGE

Comme il a été indique au début de cet article, la version de décembre 1983 est un prototype, comprenant cependant les fonctions essentielles du système. Les actions ci-après sont ensuite prévues.

- Il faudra étudier les réactions des utilisateurs. La conception de Cépage repose sur ce que nous pensons etre une bonne base ergonomique pour des systèmes interactifs, opinion confortée par des études récentes reposant sur de solides bases scientifiques [Card1983] , mais demande, bien entendu, à etre validée expérimentalement.

- Il est également prévu d'adapter le système à d'autres environnements. Cépage a été conçu pour etre portable; le choix de Pascal, de préférence à un langage orienté objets comme Simula 67 (utilisé précédemment avec succès dans la meme équipe pour réaliser des outils interactifs de qualité), était justifié par cet objectif. Il est prévu à court terme d'adapter Cépage à un environnement Unix, à la fois sur Vax et sur une station de travail SUN (à l'université de Californie); le SUN est un poste de programmation à base de 68000, possédant un écran à haute résolution ("bit-map") et une souris. Ce projet est pour nous particulièrement important, car c'est seulement dans des environnements matériels de ce niveau que des outils tels que Cépage pourront, selon nous, tenir toutes leurs promesses; nous espérons que Cépage sera également adapté à d'autres systèmes de ce type (Perq, Apollo, SM 90...).

- Il convient également d'ajouter les principales fonctions absentes du prototype, en particulier l'outil de modification du langage, et préparer des grammaires-Cépage pour les principaux langages utilisés en pratique (le prototype a été testé avec une grammaire d'un langage voisin de Pascal).

A la lumière des premières expériences, nous aurons peut-etre la réponse à quelques-unes des questions qui restent actuellement en suspens, comme celle de l'analyse syntaxique: faudra-t-il, dans une version ultérieure, ajouter un analyseur syntaxique, de facon à permettre de manipuler par Cépage des programmes existants, obtenus par d'autres moyens?

Nous espérons que la mise en service des premières versions confirmera ce que nous pensons etre le grand intéret potentiel du système actuel, et permettra d'en faire un élément essentiel d'un environnement de programmation puissant et ergonomique.

References

Abrial1980.
Jean-Raymond Abrial, Stephen A. Schuman, and Bertrand Meyer, "A Specification Language," in *On the Construction of Programs*, ed. C.A.R. Hoare and R. Perrot, Cambridge University Press, Cambridge (U.K.), 1980.

Allison1983.
R. Allison, "Syntax-Directed Program Editing," *Software, Practice and Experience*, vol. 13, pp. 453-465, April 1983.

Audin1980.
Eugène Audin, Gérard Brisson, Bertrand Meyer, and Françoise Vapné-Ficheux, "Gescran, Manuel de Référence," Atelier Logiciel 22, Electricité de France, 1980. (Fourth Edition, 1984)

Barstow1984.
David R. Barstow, "A Display-Oriented Editor for INTERLISP," in *Interactive Programming Environments*, ed. David R. Barstow, Howard E. Shrobe, Erik Sandewall, pp. 288-299, McGraw-Hill, New York, 1984.

Boehm1982.
Barry W. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs (N.J.), 1982.

Brisson1982.
Gérard Brisson, Bertrand Meyer, and Françoise Vapné-Ficheux, "Ensorcelé: Entrées et Sorties Sans Format (2ème partie)," Atelier Logiciel no. 6, Electricité de France, December 1982.

Card1983.
Stuart K. Card, Thomas P. Moran, and Allen Newell, *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale (New Jersey), 1983.

Donzeau-Gouge1981.
Véronique Donzeau-Gouge, Gérard Huet, Gilles Kahn, and Bernard Lang, "Environnement de Programmation Mentor: Présent et Avenir," in *Actes des Troisièmes Journées Francophones sur l'Informatique*, Genève, 1981.

Donzeau-Gouge.
Véronique Donzeau-Gouge, Gérard Huet, Gilles Kahn, and Bernard Lang, "Programming Environments Based on Structured Editors: The MENTOR Experience," in *Interactive Programming Environments*, ed. David R. Barstow, Howard E. Shrobe, Erik Sandewall, pp. 128-140, McGraw-Hill, New York.

Habermann1982.
Nico Habermann and others, *The Second Compendium of Gandalf Documentation*, Carnegie-Mellon University, Pittsburgh (Pa), 1982.

Hansen1971.
Wilfred J. Hansen, "Creation of Hierarchic Text with a Computer Display," ANL-7818, Argonne National Laboratory, Argonne (Ill), 1971 (Also as dissertation, Computer Science Department, Stanford University, June 1971).

1981.J.W. Lewis , "Beyond ALBE/P: Language and Neutral Form," in *Proceedings of the 5th International Conference on Software Engineering*, pp. 422-429, San Diego (Ca.), March 1981.

Kaiser1982.
Gail E. Kaiser and Elaine Kant, "Incremental Expression Parsing for Syntax-Directed Editors," Computer Science Report, Carnegie-Mellon University, October 1982.

Meyer1981.
Bertrand Meyer, "Ensorcelé: Entrées et Sorties Sans Format (1ère partie)," Atelier Logiciel no. 4, Electricité de France, April 1981. (Fourth Edition)

Meyer1982.
Bertrand Meyer, "Principles of Package Design," *Communications of the ACM*, vol. 25, no. 7, pp. 419-428, July 1982.

Meyer1983a.
Bertrand Meyer, "Towards a Two-Dimensional Programming Environment," in *Proceedings of the European Conference on Integrated Computing Systems (ECICS 82), Stresa (Italy), 1-3 September 1982*, ed. Pierpaolo Degano and Erik Sandewall, North-Holland , Amsterdam (The Netherlands), 1983.

Meyer1983b.
Bertrand Meyer and Jean-Marc Nerson, "Showing Programs on a Screen," Internal Report HI/4590-01, Electricité de France, September 1983.

Meyer1984a.
Bertrand Meyer, *A System Description Method*, Workshop on Specification Languages, to appear, Orlando (Fl.), March 1984.

Meyer1984b.
Bertrand Meyer and Jean-Marc Nerson, *Showing Programs on a Screen*, Submitted for Publication, 1984.

Mikelsons1981.
M. Mikelsons, "Prettyprinting in an Interactive Programming Environment," *SIGPLAN Notices*, vol. 16, no. 6, pp. 108-116, June 1981.

Oppen1980.
Derek C. Oppen, "Prettyprinting," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 2, no. 4, pp. 465-483, October 1980.

Schroeder1983.
Anne Schroeder, "Outils d'Analyse des Programmes sous Mentor," *GLOBULE (AFCET)*, no. 4, 1983.

Shneiderman1983.
Ben Shneiderman, "Direct Manipulation: A Step Beyond Programming Languages," *Computer (IEEE)*, vol. 16, no. 8, pp. 57-69, August 1983.

Teitelbaum1981.
Tim Teitelbaum and Thomas Reps, "The Cornell Program Synthesizer: A Syntax-Directed Programming Environment," *Communications of the ACM*, vol. 24, no. 9, pp. 563-573, September 1981.

Wilander1980.
Jerker Wilander, "An Interactive Programming System for Pascal," *BIT*, vol. 20, pp. 163-174, 1980.