Edition structuree-edition non struturee.Cooperation et complementarite

B. Melese

▶ To cite this version:

B. Melese. Edition structuree-edition non struturee.Cooperation et complementarite. RR-0253, INRIA. 1983. inria-00076305

HAL Id: inria-00076305
https://inria.hal.science/inria-00076305

Submitted on 24 May 2006

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.

INRIA

CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 253

ÉDITION STRUCTURÉE - ÉDITION NON STRUCTURÉE

COOPÉRATION ET COMPLÉMENTARITÉ

Bertrand MÉLÈSE

Novembre 1983

Edition Structurée - Edition Non Structurée
Coopération et Complémentarité

Structured Editing - Unstructured Editing
Cooperation and Complementarity

Bertrand Mélèse
I.N.R.I.A
Domaine de Voluceau - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex
Tel: (3) 954-90-20

Résumé

Nous soutenons que dans un environnement de programmation, l'édition structurée et l'édition non structurée ont chacune un rôle spécifique à jouer. Ces deux modes d'édition doivent donc coexister et coopérer de façon étroite. L'argumentation s'appuie sur une expérience d'utilisation semi-intégrée du système Mentor avec un éditeur de textes. Une application à l'édition structurée de documents techniques est présentée: le système Mentor-Rapport.

Abstract

We claim that in a programming environment structured editing and unstructured editing both have a specific role to play. These two editing modes must be available to the user of such an environment and must cooperate in a smooth way. Our claim is based on a practical experience using a running interface between the Mentor system and a text editor. The Mentor-Rapport system is presented as an application to the edition of technical reports.

PAPIER RÉCUPÉRÉ ET RECYCLÉ

Edition Structurée - Edition Non Structurée

Coopération et Complémentarité

Bertrand Mélèse
I.N.R.I.A.
Domaine de Voluceau - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex
Tel: (3) 954-90-20

1. Introduction

On peut, en première approche, classer les composantes de la plupart des formalismes en deux groupes. Les composantes qui ont une structure syntaxique forte reflétant de près la sémantique: c'est par exemple le cas des instructions d'un langage de programmation. Les composantes dont la structure syntaxique est beaucoup plus libre vis à vis de la sémantique, telles le texte des commentaires d'un programme ou le texte d'un paragraphe dans un document en langage naturel.

Alors qu'il est naturel de manipuler les composantes à structure syntaxique forte avec des systèmes dirigés par la syntaxe, ces systèmes sont mal adaptés à la manipulation des composantes à structure syntaxique faible. Pour ces dernières il semble raisonnable de s'en tenir aux méthodes d'édition de textes non structurés, tout en cherchant à profiter des outils les plus performants en ce domaine.

Dans ce papier nous developpons l'idée que l'utilisateur d'un environnement de programmation devrait toujours avoir le choix entre l'édition structurée et l'édition non structurée selon le type de tâche qu'il se propose d'accomplir. La faisabilité d'un système dans lequel ce choix est toujours possible est démontrée par une implémentation expérimentale effectuée autour du système Mentor. La preuve de l'utilité d'un tel système

est son emploi systématique par les utilisateurs qui ont accès à cette version de Mentor.

Insistons sur le fait que, dans notre conception d'un environnement de programmation, le fait de donner le choix entre édition structurée et édition non structurée ne remet pas du tout en cause la supériorité de la **représentation structurée** des objets par rapport à leur représentation textuelle. Ce choix n'existe que dans le mode d'interaction entre l'utilisateur et le système et n'a aucune incidence sur la représentation interne des objets: à la fin d'une session d'édition en mode texte (édition non structurée) le système reconstitue, par analyse syntaxique et construction d'arbres, la représentation structurée des objets.

L'idée de mélanger les deux types d'éditions existe, sous une forme simplifiée, dans le *Cornell Program Synthesizer* (C.P.S) [TEIT 81]. Dans ce système, les programmes sont édités structurellement sauf pour les expressions, lesquelles sont éditées et **conservées** sous forme de texte, les concepteurs du C.P.S ayant considéré la manipulation structurée des expressions comme pénible et peu profitable.

Notre approche diffère de celle du C.P.S par un souci de généralisation et d'unification des concepts:

- Dans Mentor, la représentation interne est toujours l'arbre de syntaxe abstraite.

- Nous considérons la distinction, faite dans le C.P.S, entre les expressions et les autres constructions d'un langage comme arbitraire. En réalité, cette distinction provient du fait que le C.P.S est un système sans analyseur syntaxique. Avec Mentor, disposant d'un analyseur syntaxique multi-points d'entrées, nous pouvons être plus ambitieux et fournir le choix entre édition structurée et édition non structurée pour tous les composants du formalisme manipulé.

- Nous pensons qu'il est préférable que l'utilisateur puisse choisir un éditeur de textes qui lui est familier plutôt que de lui en imposer un autre, spécifique au système. Nous nous sommes donc appliqués à rendre

très facile l'adaptation d'un nouvel éditeur de textes au système.

La section 2 de ce papier est un bref exposé des principales caractéristiques du système Mentor. Dans la section 3 nous exposons les raisons pour lesquelles la possibilité d'édition non structurée, à l'intérieur d'un environnement de programmation nous paraît importante. Dans la section 4 nous expliquons la communication entre Mentor et un éditeur de textes. Enfin, dans la section 5, nous décrivons une application qui utilise intensivement la coopération entre les 2 types d'édition: le système de préparation de documents techniques *Mentor-Rapport*.

2. Principales caractéristiques du système Mentor

Le système Mentor [DON 75, DON 83, DON 84] est un environnement de programmation basé sur le concept d'édition structurée guidée par la syntaxe du langage auquel les objets manipulés appartiennent. Il existe d'autres systèmes basés sur ce concept: Gandalf [FEI 81, HAB 79], le "Cornell Program Synthesizer" [TEIT 81, REPS 82] et Adèle [MOSS 82]. Ces systèmes et Mentor partagent les idées suivantes:

- Représentation arborescente des objets manipulés. La forme textuelle des objets est calculée à partir de l'arbre par un processeur nommé *décompilateur*.

- Utilisation d'une *syntaxe abstraite* pour définir les arbres légaux du formalisme édité et pour contrôler que ces arbres restent légaux au cours des transformations qui leur sont appliquées pendant une session d'édition. Un arbre est *légal* si le texte qui en résulte par décompilation est une phrase légale du formalisme.

- Existence d'un ensemble de commandes interactives pour la manipulation des arbres de syntaxe abstraite.

Les principaux points par lesquels Mentor se distingue des autres systèmes sont les suivants:

- Mentor est *programmable*. Le langage de commande et un vrai langage de programmation appelé **Mentol** [DON 80]. Ce langage peut être utilisé pour développer des bibliothèques de programmes de manipulations et de transformations d'arbres. Ces programmes peuvent ensuite être invoqués interactivement comme des nouvelles commandes [MEL 80, MEL 81].

- Mentor est *indépendant du langage*. Les données spécifiques à un langage sont rangées dans des tables externes au système. Pour passer d'un langage à un autre, le système charge les tables du nouveau langage et toutes les primitives du système sont alors guidées par ces nouvelles données. Un langage de spécification, **Métal** [KAHN 83, MEL 82], permet de définir les langages que l'on souhaite introduire sous Mentor.

- Mentor est *multi-langages*. Il est possible de manipuler simultanément des arbres appartenant à des formalismes différents. Par les mécanismes d' *annotations* et de *portes* il est également possible de mélanger plusieurs formalismes dans un même arbre [DON 83].

- Mentor est un système *ouvert*. Tous ses composants sont accessibles à partir de programmes utilisateurs, écrits en Pascal, PL1 ou C, à travers une interface standard. Un utilisateur peut donc ajouter de nouveaux processeurs au système ou même utiliser Mentor comme un environnement de base pour l'implémentation d'un système spécialisé [MIG 83, SCH 83].

3. Le rôle de l'édition non structurée dans Mentor

Dans certaines situations, l'édition structurée se révèle peu performante. Nous détaillons dans cette section les cas les plus fréquents dans lesquels un utilisateur veut s'affranchir des contraintes imposées par la représentation structurée lorsqu'il developpe un programme sous Mentor.

3.1. Edition des composantes non structurées d'un langage

Tous les langages de programmation contiennent des composantes non structurées (commentaires, chaines de caractères, identificateurs) qu'il est naturel d'introduire et de modifier avec un éditeur de texte.

Le langage **Rapport** , dont il est question dans la section 5, est un exemple de formalisme dans lequel une large part est faite aux composantes non structurées et qu'il serait pénible, voir impossible, d'éditer de façon purement structurée.

3.2. Saisie de nouveau code

La saisie de nouveau code directement sous Mentor, c'est à dire en mode structuré, n'est pas très agréable pour plusieurs raisons:

- Le code entré à la console est analysé ligne par ligne. Les possibilités d'édition locale du texte tapé sont donc limitées à la ligne en cours de saisie.

- En cas d'erreurs détectées par l'analyseur syntaxique, la récupération d'erreurs qui s'ensuit n'est pas toujours facile à prévoir ce qui rend

acrobatique la terminaison propre de l'entrée en cours.

- Enfin, lorsque des erreurs non récupérables ont été detectées, des parties plus ou moins importantes du texte entré peuvent être perdues.

Il est donc plus agréable d'avoir affaire à un éditeur de textes pour saisir la totalité des données puis d'envoyer celles-ci en un coup à Mentor. En cas d'erreurs il est alors possible de récupérer le texte original pour le corriger avant de l'envoyer à nouveau à Mentor. L'efficacité et la facilité d'un tel mode de saisie dépend uniquement de la qualité de la communication entre Mentor et l'éditeur de textes utilisé.

Un autre mode de saisie possible, utilisé systématiquement dans Gandalf et dans le C.P.S est la construction explicite, par le programmeur, des noeuds de l'arbre de syntaxe abstraite à l'aide de commandes prédéfinies. Seules les unitées lexicales du langage, qui seront les feuilles de l'arbre (plus les expressions dans le cas du C.P.S) sont alors saisies en totalité.

Ce mode à l'avantage de dispenser le programmeur de la frappe des mots clés du langage et facilitera donc la tâche d'un programmeur débutant ou peu familier avec le langage utilisé. Ce mode existe dans Mentor (construction de l'arbre à l'aide des schémas prédéfinis) mais nous le trouvons très lourd: il revient en effet à construire à la main l'arbre en ordre préfixe ce qui est rapidement insupportable pour un programmeur expérimenté. Dans Mentor, ce mode ne sera utilisé que ponctuellement, éventuellement en liaison avec l'éditeur de textes.

Notons encore une fois ici que le choix de ce mode de saisie dans Gandalf et dans le C.P.S provient du fait que ces systèmes n'ont pas d'analyseur syntaxique.

3.3. Edition de structures profondes

Le cas des expressions d'un langage de programmation, déjà évoqué dans l'introduction, est un exemple de structure pour laquelle une représentation textuelle courte peut engendrer une représentation arborescente profonde. En Mentor-Pascal par exemple, l'expression A+(D mod 13 -B * (C-1) div 2) est représentée par l'arbre de la figure 1. Un utilisateur jugera en général plus confortable d'éditer une expression en mode texte plutôt que directement sur l'arbre qui la représente.

4. Réalisation de la communication entre Mentor et un éditeur de textes

La communication entre Mentor et un éditeur de textes est réalisée grâce au décompilateur et à l'analyseur-constructeur d'arbres (figure 2).

Rappelons que le *décompilateur* est le processeur qui crée le texte d'un programme à partir de sa représentation arborescente tandis que *l'analyseur-*

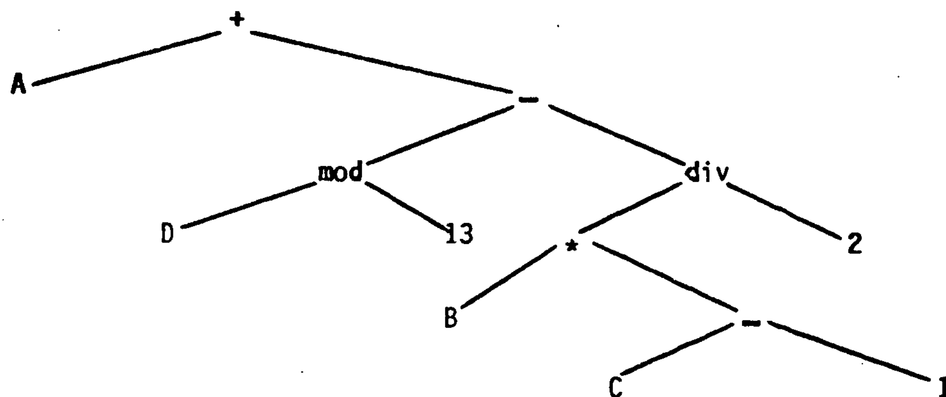

figure 1: Représentation d'une expression en Mentor-Pascal

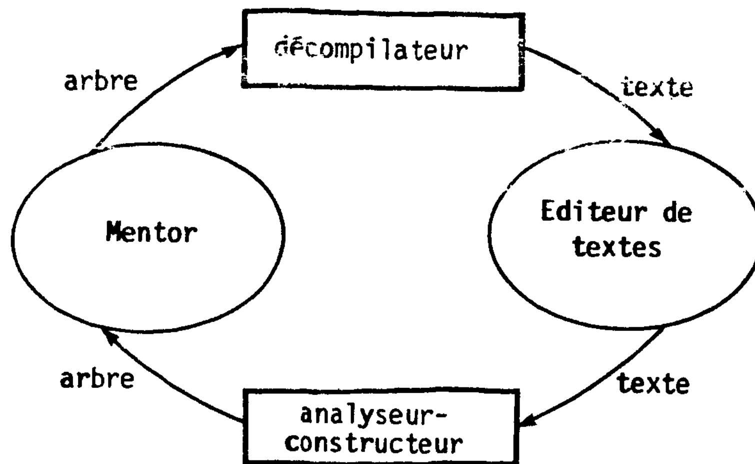

figure 2: Communication Mentor-Editeur de textes

constructeur (d'arbres) est le processeur qui, à partir du texte d'un programme crée sa représentation arborescente. Le décompilateur est utilisé systématiquement pour visualiser les programmes manipulés tandis que l'analyseur-constructeur est utilisé lors de l'entrée (interactive ou non) de programmes ou de fragments de programmes.

Etant donnée l'existence de ces deux processeurs le principe de la communication est très simple. Pour envoyer un fragment de programme de Mentor vers l'éditeur de textes il suffit de créer, par décompilation, le texte correspondant puis d'envoyer ce texte à l'éditeur de textes. Cet envoi est actuellement réalisé par l'intermédiaire d'un fichier temporaire. Il sera réalisé par une "pipe" dans la version de Mentor sur Unix. De la même façon, pour envoyer un fragment de programme de l'éditeur de textes vers Mentor on envoie le texte correspondant dans l'analyseur-constructeur lequel crée l'arbre correspondant qui est ensuite raccroché à l'arbre principal par les primitives standard de Mentor.

Ce mode de communication a les deux avantages suivants:

- Etant réalisé à travers deux processeurs standards de Mentor, il est possible de transmettre n'importe quel fragment de programme tirant parti du fait que l'analyseur-constructeur est multi-points d'entrées.

- Il est adaptable facilement à tous les éditeurs de textes. Il est aujourd'hui opérationnel avec les éditeurs TED et EMACS dans la version de Mentor sur Multics et avec les éditeurs ED, EMACS et EMIN dans la version Vax-Unix.

Du point de vue de l'utilisateur de Mentor, cette communication est mise en oeuvre, dans le sens Mentor -> éditeur de textes, par une commande unique (*édit*). Cette commande est invoquée de l'intérieur de Mentor et rend la main dans l'éditeur de textes avec le texte résultant de la décompilation du sous arbre courant chargé et prêt à être édité. Dans l'autre sens, le simple fait de sortir de l'éditeur de textes, par la commande de sortie normale de celui-ci, rend la main sous Mentor, l'arbre courant étant remplacé par celui résultant de l'analyse du texte tel qu'il a été laissé dans l'éditeur de textes.

4.1. La communication Mentor <-> Emacs

Dans le cas de la communication entre Mentor et Emacs, une interface utilisateur sophistiquée a été réalisée. Rappelons que Emacs est un éditeur de texte plein écran utilisable sur tout terminal ayant des possibilités d'adressage du curseur et des fonctions d'insertion et de destruction de lignes et de caractères.

L'interface réalisée dans ce cas sépare l'écran en deux fenêtres, une pour Emacs et une pour Mentor. Du point de vue de l'utilisateur, la commande ***édit*** deplace le texte correspondant au sous arbre courant dans la fenêtre Emacs et rend la main dans celle-ci. La sortie de Emacs a l'effet inverse: l'arbre correspondant au texte édité devient l'arbre courant de la session Mentor.

Seule la commande de sortie de Emacs a été modifiée pour les besoins de cette communication. En effet, cette commande revient à Mentor tout en laissant Emacs actif. Ceci est réalisé grâce au mécanisme d'empilement des processus sur Multics et grâce à la communication par "pipes" sur Unix. Le fait de laisser Emacs actif est important puisque cela évite de payer son temps de chargement à chaque appel.

Notons que cette interface n'est transportable que sur les terminaux qui supportent Emacs et sur lesquels il est possible de diviser l'écran en deux fenêtres. Notons aussi que l'utilisateur peut, à tout moment, agrandir la fenêtre Mentor à tout l'écran sans pour autant tuer le processus Emacs et donc sans coût supplémentaire au prochain appel.

4.2. Problèmes d'intégration et développements futurs

Le principal désagrément, lors de l'utilisation de la communication Mentor-Editeur de textes, provient du fait que les deux systèmes sont totalement disjoints ce qui entraine un mode de travail haché: il est nécessaire d'appeler explicitement l'éditeur et il est impossible de charger, sous l'éditeur, le texte correspondant à un autre sous arbre sans transiter par Mentor.

Pour améliorer cette interface, en particulier dans le cas Mentor-Emacs, il faudrait ouvrir la fenêtre Emacs autour du texte qui correspont au sous arbre à modifier. En effet, ce texte, décompilé par Mentor, est déjà sur l'écran. Le seul fait de demander l'édition en mode texte d'un sous arbre rendrait alors actives les commandes de l'éditeur de textes sur la représentation textuelle de ce sous arbre. Nous comptons réaliser une interface de ce type dès que nous disposerons d'écrans à haute résolution.

5. Mentor-Rapport

Rapport est un langage de description de documents. En *Rapport* , un document est défini de façon naturelle en termes de chapitres, sous-chapitres, paragraphes, bibliographie etc.... Aucune tentative n'est faite pour modéliser la structure du langage naturel: les titres, paragraphes ainsi que toutes les composantes textuelles sont considérées, en première approche, comme des atomes du langage.

Mentor-Rapport est un système de manipulation de documents structurés selon la syntaxe du langage *Rapport*. Nous décrivons ici succintement Mentor-Rapport. Pour plus de détails le lecteur se reportera au manuel d'utilisation de ce système [MEL 83].

5.1. Structure d'un document en Mentor-Rapport

Suivant les principes de base du système Mentor, un document, en Mentor-Rapport, est représenté par un arbre qui obéit aux règles de la syntaxe abstraite du langage Rapport. Cette syntaxe abstraite est décrite complètement dans [MEL 83] et nous n'en donnons ici qu'une idée générale et informelle.

En Mentor-Rapport un document est composé de **descripteurs** utilisés par le système, d'un **en-tête** dans lequel figurent le titre, les noms des auteurs, et leurs addresses, d'une liste de **chapitres** et d'une bibliographie. Un chapitre est composé d'un descripteur, d'un titre et d'une liste de composants, chaque composant pouvant être un paragraphe ou un (sous-) chapitre. La bibliographie est une liste de **références** composées d'une étiquette, par laquelle elle sera nommée dans le texte, des noms des auteurs, du titre de l'article et des indications nécessaires pour se procurer cet article.

Les paragraphes, titres, noms des auteurs dans la bibliographie sont des exemples d'atomes du langage Rapport et donc de feuilles de l'arbre qui représente un document en Mentor-Rapport.

Lorsque l'utilisateur décide d'éditer, ou de créer un tel atome, Mentor-Rapport appelle automatiquement l'éditeur de textes par le mécanisme de communication décrit au chapitre précédent. Cet appel automatique correspond à l'idée de *portes* [DON 83] qui peut se résumer en disant que certains atomes d'un formalisme, ici les morceaux de texte, sont des objets dont la structure appartient à un autre formalisme que le formalisme principal et doivent donc être manipulés par d'autres primitives. En Mentor-Rapport, les morceaux de texte sont structurés comme des suites de caractères et les primitives pour les manipuler sont des éditeurs de textes.

L'interface entre Mentor et les éditeurs de textes est donc améliorée dans le cas de Mentor-Rapport par le fait que ce système reconnaît les sous-arbres (atomes) qui sont toujours édités textuellement. Dans la plupart des cas Mentor-Rapport est donc capable d'appeler l'éditeur de texte automatiquement dès qu'une telle structure est atteinte.

5.2. L'environnement Mentor-Rapport

Dans ce chapitre nous décrivons succintement quelques-unes des commandes disponibles pour faciliter la tâche d'écriture de documents techniques depuis la saisie jusqu'à la mise en page sur un périphérique classique (imprimante) ou spécialisé (photo-composeuse, imprimante electrostatique). La plupart de ces commandes fonctionnent par questions-réponses et sont accessibles aux utilisateurs non spécialistes de Mentor. Toutes les commandes standard de Mentor sont disponibles en Mentor-Rapport. Un système de *marques* rend la plupart d'entre elles utilisables par les débutants.

5.2.1. Exemples de commandes de Mentor-Rapport

5.2.1.1. Document_français (df)

Création d'un nouveau document en français. Cette procédure est la première à appeler lorsque l'on commence la rédaction d'un document en français. Elle construit l'en-tête du document en posant des questions.

5.2.1.2. Créer

Procédure de création d'une partie de document: chapitre paragraphe, ou bibliographie. Le choix entre ces possibilitées est déterminé par la réponse à la première question posée. L'emplacement de la partie créée dépend de la position au moment de l'appel pour les chapitres et les paragraphes. La bibliographie est toujours positionnée à la fin du document, avec concaténation à la bibliographie déjà existante si il y en a une.

La demande de création d'un paragraphe a pour effet d'appeler l'éditeur de textes actif. Le paragraphe est entré sous cet éditeur. Il est integré au document lors du retour de l'éditeur.

5.2.1.3. Section (chapitre)

Remonte au chapitre englobant s'il en existe un. Ne bouge pas s'il n'y en a pas.

5.2.1.4. Psuiv, Ppre, Pder

La procédure **psuiv** va au paragraphe suivant du document. Elle ne bouge pas si il n'y en a pas. Cette procédure peut, si nécessaire, changer de chapitre pour trouver le paragraphe suivant.

La procédure **ppre** va sur le paragraphe précédent. Elle ne bouge pas si il n'y en a pas.

La procédure **pder** va sur le dernier paragraphe du document courant. Elle remonte au début si il n'y a aucun paragraphe dans ce document.

5.2.1.5. Paragraphes_courts (pc), Paragraphes_longs (pl)

Les procédures **pc** et **pl** sont des procédures de changement d'état: elles influent sur le comportement ultérieur du système.

La procédure **pc** met le système dans l'état *Paragraphes en forme courte* ce qui signifie que les paragraphes de plus de 4 lignes seront abrégés par leurs 2 premières lignes suivies de points de suspension lors de leur affichage à l'écran. Cette procédures fournit un moyen de voir les atomes paragraphes sous une forme abrégée et vient donc compléter le mécanisme standard d'abréviation, toujours présent dans Mentor, par lequel un arbre, ici un document, peut être visualisé à différents niveaux de détails.

La procédure **pl** met le système dans l'état *Paragraphes en forme longue*. Dans cet état les paragraphes sont toujours affichés en entier.

5.2.1.6. Chercher

Cette commande demande l' *identification* d'un chapitre et va sur le chapitre correspondant. Elle ne bouge pas si ce chapitre n'existe pas.

L' *identification* d'un chapitre est un identificateur demandé par la procédure **créer** lorsqu'on demande la création d'un chapitre. Cet identificateur n'est qu'une marque destinée à faciliter la recherche: il serait en effet difficile et peu agréable de repérer un chapitre par son titre, celui-ci

pouvant être long à taper. Cet identificateur n'apparaîtra pas dans la version finale du document.

5.2.1.7. Bib, Tribib

La commande **bib** va sur la bibliographie du document si elle existe.

La commande **tribib** trie les références bibliographiques sur leurs champs *identification* et *date*. Ces champs sont les composants de l'étiquette de chaque référence. Ils sont demandés par la commande **créer** au cours de la création de la bibliographie.

5.2.1.8. Plan

La commande **plan** présente à l'écran la table des matières du document avec la numérotation des chapitres. Celle-ci pourra être intégrée au document lors de sa mise en page définitive. Cette commande procède par simple traversée de l'arbre. Les numéros des chapitres et sous-chapitres sont déduits de leur profondeur dans l'arbre. La numérotation est recalculée à chaque appel de la commande et est donc toujours à jour quelque soient les modifications faites au document (échange, fusion, destruction de chapitres ...).

5.2.2. Mise en page des documents

Une des principales fonctionnalitées de Mentor-Rapport est la commande **compose** qui "compile" l'arbre représentant un document dans une forme acceptable par un système de composition de textes tel que Compose [MUL 79] sur Multics ou Troff [KER 78] sur Unix. L'utilisateur n'a donc pas besoin d'apprendre à se servir des systèmes de composition de textes, et est assuré de la cohérence de la mise en page de son document. Le compilateur fonctionne par décompilation de l'arbre et insertion de macro-commandes du système de composition dans le texte produit. Pour modifier la forme standard de composition il suffit donc de modifier les macro-commandes ce qui est à la portée de tout utilisateur confirmé.

La forme standard sous laquelle les documents sont composés est illustrée par le présent papier. Les changements de fontes sont effectuées par des commandes internes à Mentor-Rapport, indépendantes du système de composition

utilisé.

Actuellement, deux compilateurs sont opérationnels. Le premier est adapté au système Compose de Multics et permet de sortir les documents sur une photo-composeuse. Le présent papier a été composé avec le compilateur adapté au système Troff de Unix et a été sorti sur une imprimante Versatec.

5.3. Documents génériques et formats imposés

Dans cette section nous rappelons les notions de *schémas* et de *métavariables* utilisées dans Mentor. Nous montrons ensuite des applications dans le cadre de Mentor-Rapport.

5.3.1. Schémas et Métavariables

Nous appelons *schéma* un arbre incomplet, c'est à dire un arbre dans lequel un ou plusieurs sous arbres sont remplacés par des *métavariables*.

Les schémas sont utilisés pour la recherche associative dans les arbres ("tree pattern matching"). Un arbre est une *instance* d'un schéma si il ne diffère de ce schéma qu'aux emplacements des métavariables. La recherche d'un schéma dans un arbre trouve le premier sous arbre qui est une instance du schéma. En cas de succès de la recherche, les métavariables sont liées aux sous arbres qui leur correspondent dans le sous arbre trouvé. Ces liaisons peuvent ensuite être utilisées pour *instancier* d'autres schémas c'est à dire remplacer dans ceux-ci les métavariables par les arbres auquelles elles ont été liées par une recherche précédente. Au cours de l'instanciation d'un schéma, les métavariables libres ne sont pas modifiées. En Mentor, les métavariables sont des identificateurs précédés du caractère "$".

Exemple:

En Mentor-Pascal, la recherche du schéma $A*($B+$C) dans l'arbre dont le texte est A[I] := (X+1)*(Y+Z) trouvera le membre droit de l'affectation et liera les métavariables de la façon suivante:

```
$A  —> X+1
$B  —> Y
$C  —> Z
```

L'instanciation du schéma ($A*$B)+($A*$C) avec ces liaisons produit l'arbre dont le texte est ((X+1)*Y)+((X+1)*Z).

Les schémas et les instanciations sont intensivement utilisés dans Mentor, souvent à l'insu de l'utilisateur. Nous nous intéressons ici à leur emploi pour la description de programmes génériques (documents génériques en Mentor-Rapport) et pour la construction, à partir de ceux-ci, de programmes particuliers par instanciation.

5.3.2. Un exemple en Mentor-Pascal

L'absence de tableaux dynamiques en Pascal interdit d'écrire une procédure unique pour multiplier deux matrices. Par contre, toutes les procédures de multiplication de matrices sont calquées sur le même modèle: seuls les types des matrices et les indices des boucles changent.

On peut donc écrire, en Mentor-Pascal, un modèle de procédure de multiplication de matrices, que nous appelerons *procédure générique* de multiplication de matrices, dans laquelle les bornes des tableaux et les indices des boucles sont des métavariables. Pour introduire une procédure de multiplication de matrices dans un programme, il suffit de créer les liaisons adéquates sur les métavariables de la procédure générique, par exemple par des recherches associatives dans les déclarations des matrices, puis d'instancier ces métavariables. Le travail de recherche et d'instanciation est fait par une commande **Mentol** et est donc complètement caché à l'utilisateur qui appelle une commande unique pour utiliser une procédure générique existant dans la bibliothèque. Dans le cas de la multiplication des matrices, la commande **Mentol** vérifie que les matrices sont multipliables l'une par l'autre en calculant les dimensions des tableaux qui les représentent. Des exemples de cette utilisation des schémas sont détaillés dans [DON 84].

5.3.3. Documents génériques en Mentor-Rapport

On peut, en Mentor-Rapport créer des bibliothèques de documents génériques tels que des lettres, contrats, annonces de conférences, de la même façon qu'en Mentor-Pascal il est possible de créer des bibliothèques de procédures génériques. Les documents génériques sont des schémas matérialisant simplement le squelette du document. Pour créer un document particulier avec ce format, un système de questions-réponses parcours ce squelette et demande à l'utilisateur ce qu'il faut mettre dans les trous, c'est à dire à la place des métavariables. Une simple instanciation crée finalement le document voulu.

Grâce au système d'annotations de Mentor [DON 83] on peut attacher des instructions Mentol dans un arbre Mentor-Rapport et donc créer des documents génériques qui contiennent le système de questions-réponses qui leur est adapté. Cette technique est utilisée par la commande **créer** de l'environnement Mentor-Rapport pour la création de la bibliographie.

6. Conclusion

L'importance de l'existence d'édition textuelle dans un environnement basé sur le principe de l'édition structurée est largement démontrée par l'utilisation systématique de la liaison Mentor <-> Editeur de textes, et plus particulièrement Mentor <-> Emacs, depuis sa mise en service.

L'agrément de cette liaison provient pour une large part de l'existence d'un analyseur-constructeur d'arbres multi-points d'entrées ce qui permet d'éditer en mode texte n'importe quel fragment de programme et donc de faire de la saisie de façon agréable et efficace.

Depuis sa création fin 1982, Mentor-Rapport est systématiquement utilisé par la communauté Mentor de l'INRIA et de nombreux documents, dont

plusieurs thèses, ont déjà été produits en Mentor-Rapport. Certains utilisateurs ont adapté le format standard de composition à leurs besoins propres sans interférences avec les autres utilisateurs.

Mentor-Rapport doit être considéré aujourd'hui comme une preuve de faisabilité et d'utilité d'un système de manipulation de documents. La structure utilisée est simpliste et doit s'enrichir pour utiliser plus largement les possibilités de Mentor comme les annotations et la manipulation de plusieurs langages simultanément. Ces deux fonctionnalitées de Mentor conduiront naturellement Mentor-Rapport à être un système d'édition de documents multi-formalismes.

Bibliographie

Don 75

V. Donzeau-Gouge, G. Kahn, G. Huet, B. Lang, J.J. Levy, *A structure oriented program editor: a first step toward computer assisted programming*, International Computing Symposium North_Holland Publishing Co.

Don 80

V. Donzeau-Gouge, G. Huet, G. Kahn, B. Lang, *Programming environment based on structured editors: The Mentor experience*, Rapport de Recherche No. 26, INRIA, Juillet 1980

Don 83

V. Donzeau-Gouge, G. Kahn, B. Lang, B. Mélèse, *Outline of a tool for document manipulation*, IFIP, septembre 1983, Paris

Don 84

V. Donzeau-Gouge, B. Lang, B. Mélèse, *Practical Applications of a Syntax Directed Program Manipulation Environment*, Soumis pour publication, disponible auprès des auteurs

Fei 81

P. H. Feiler, R. Medina-Mora, *An incremental programming environment*, IEEE trans. on soft. Eng. SE-7, No 5, Sept. 81, 472-481

Hab 79

A. N. Habermann, *The Gandalf Research Project*, Computer Science Research Rewiew 1978-79, pages 28-35

Kahn 83

G. Kahn, B. Lang, B. Mélèse, *Metal: a formalim to specify formalisms*, A paraître dans Science of Computer Programming, North Holland, 1983

Ker 78

B. W. Kernighan, M. E. LESK, J. F. Ossanna, Jr, *Document Preparation*, The Bell System Technical Journal, July-August 1978, Vol. 57, No. 6, Part 2

Mel 80

B. Mélèse, *Manipulation de programmes Pascal au niveau des concepts du langage*, Thèse de 3ieme cycle, Université Paris 11 Orsay, 1980

Mel 81

B. Mélèse, *Mentor: l'environnement Pascal*, Rapport Technique No 5, I.N.R.I.A., Octobre 1981

Mel 82

B. Mélèse, *Métal, un langage de spécification pour le système Mentor*, Technique et Science Informatique (AFCET), Vol. 1 No 4, Juillet-Aout 1982

Mel 83

B. Mélèse, *Mentor Rapport: Manipulation de textes structurés sous Mentor*, Rapport Technique No. 23, INRIA, Avril 1983

Mig 83

V. Migot, *Un Pascal Modulaire sous Mentor*, Thèse de 3ième cycle, université Paris 11, sept. 1983

Moss 82

J. Mossiere, J. Raymond, Y. Rouzaud, *Représentation interne et manipulation des programmes dans Adèle*,. Colloque Génie Logiciel AFCET, Paris, Janvier 1982

Mul 79

Wordpro New User's Guide, Documentation of the Multics System.

Reps 82

T. Reps, *Generating Langage Based Environments*, Technical Report 82-514,Cornell University, Ithaca, NY, August 82

Sch 83

A. Schroeder, *Integrated Program Measurement and Documentation Tools*, Rapport de Recherche No 227,INRIA, Soumis pour publication

Teit 81

T. Teiltelbaum, T. Reps, *The Cornell Program Synthesizer: A syntax directed programming environment*, Communication of the ACM, vol. 24, no. 9, September 81, 563-573

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique