Interface pour le calcul formel
Stage de DEA

présenté par

Stéphane Lavirotte

stage effectué à

INRIA Sophia-Antipolis
2004 Route des Lucioles
06902 Sophia Antipolis CEDEX

Encadreurs :

Stéphane Dalmas
Marc Gaëtano

Rapporteur :

Erick Gallésio

Novembre 1994 - Juin 1995

Remerciements

Je tiens à remercier :

Mes encadreurs de stages Stéphane Dalmas et Marc Gaëtano pour tous leurs conseils et le temps qu'ils m'ont consacré.

Olivier Arsac pour toutes ses explications sur EMATH, pour le temps qu'il a bien voulu m'accorder, pour ses conseils et sa bonne humeur.

L'équipe SAFIR qui m'a accueilli et donné les moyens de mener à bien ce travail.

Catherine Ossola et Jérôme Desousa mes collègues de bureau, pour leur bonne humeur et l'accueil qu'ils m'ont fait.

Alain Sausse pour ses conseils sur LaTeX.

Les membres de l'équipe KOALA (projet Bull à l'INRIA) pour leurs conseils et leur aide sur leurs produits qu'ils m'ont permis d'utiliser (Klone et Klm) : Jean-Michel Léon, Colas Nahaboo, Arnaud Le Hors, Cédric Beust, Philippe Kaplan.

Table des matières

Introduction

Un système de calcul formel est un logiciel capable de résoudre des problèmes mathématiques de manière symbolique. On peut le voir comme une bibliothèque de composants logiciels manipulant des objets mathématiques et un ensemble d'outils permettant de les lier et de les exécuter, dont une interface qui permet d'intéragir avec le moteur de calcul.

Dans ce rapport, nous nous intéresserons à la conception et à la réalisation d'interface pour des systèmes de calcul formel.

Durant ces dernières années, les outils pour le calcul formel ont énormément gagné en puissance et en rapidité. Ces systèmes sont maintenant largement disponibles pour les utilisateurs d'ordinateurs personnels. Ainsi, de plus en plus d'utilisateurs non spécialistes se servent d'outils tels que *Maple, Mathematica, Reduce...* Ces utilisateurs étant souvent occasionnels, le temps d'apprentissage de ces logiciels doit être minimal, et la qualité de l'interface est un critère important. La plupart des systèmes actuels ont été conçus entre la fin des années soixante et le début des années quatre-vingts, ce qui peut expliquer le déficit en ergonomie des interfaces. Mais, en tout état de cause, il reste difficile de concevoir de telles interfaces.

Un des problèmes majeurs de ces interfaces pour le calcul formel est que l'on ne dispose pas d'une interface qui soit indépendante du système de calcul auquel elle est associée. Ceci nuit à une certaine productivité ainsi qu'au confort d'utilisation. L'utilisateur doit faire un effort d'apprentissage et se plier aux méthodes de manipulation fournies par l'interface.

Puisqu'on ne peut pas avoir une interface qui soit universelle pour tous les utilisateurs, il est préférable que chaque utilisateur puisse paramétrer celle-ci en fonction de ses propres besoins. On doit donc pouvoir modifier, de manière simple, le rendu visuel mais aussi les mécanismes de saisie des formules, de manipulation ...

Par exemple, pour des utilisateurs manipulants de gros polynômes (plusieurs centaines de termes), il est impossible de les exploiter avec une interface classique qui se contente d'afficher tous les termes. On doit donc pouvoir paramétrer l'interface pour n'avoir que l'affichage des termes pertinents. Dans le même domaine, pour une saisie répétitive d'un certain type de formule, comme les polynômes, on peut vouloir faire une optimisation de certains mécanismes ; par exemple, après avoir saisi l'exposant d'une variable, on retourne automatiquement au niveau de la variable pour la saisie de l'opérateur suivant.

Pour que l'interface puisse s'adapter aux différents utilisateurs, il faut que celle-ci dispose d'un certain degré de paramétrabilité.

Ce stage a pour but de mettre en évidence les différentes caractéristiques que devrait avoir une interface ainsi que des mécanismes adéquats pour la saisie, la manipulation de formules et l'organisation des calculs... Ceci conduira à la réalisation d'une interface expérimentale. Pour cela, on utilisera un serveur (EMATH) qui réalise les opérations de base pour l'édition de formules. Nous avons choisi EMATH afin de s'abstraire complètement des problèmes liés à l'affichage des formules. Cette interface pourra se connecter à différents systèmes de calcul formel de manière transparente grâce à un autre outil le(CENTRAL CONTROL) développé au sein du projet SAFIR.

Grâce à cette organisation, l'interface sera largement indépendante du système de calcul formel auquel (ou auxquels) elle sera connectée ; ceci ayant pour avantage d'avoir toujours la même interface quelque soit le système utilisé.

Dans le chapitre 1, nous tenterons de faire une classification des différents modèles d'interfaces pour le calcul formel et de mettre en évidence les points forts et les points faibles des interfaces existantes ainsi que les problèmes que l'on peut rencontrer dans leur conception.

Dans le chapitre 2, nous proposerons un autre modèle d'interface et un début de spécification.

Dans le chapitre 3, nous ferons une présentation du composant EMATH qui a servi à la réalisation d'un prototype et nous justifierons ce choix.

Le chapitre 4 présentera les choix que nous avons fait pour résoudre les problèmes exposés au chapitre 1 et la bibliothèque de fonctions qui a été mise en place.

Le chapitre 5 mettra en évidence les différents atouts du modèle d'interface proposé et présentera le prototype réalisé.

Chapitre Premier

L'existant et les problèmes

1.1 Historique

Le problème des interfaces pour la manipulation de formules mathématiques a été soulevé très tôt. En effet, en 1963, Marvin Minsky [Min63] suggère déjà quelques directions générales pour la manipulation de formules sur un écran d'ordinateur.

C'est aussi à partir de 1963 qu'apparaîssent les premières interfaces pouvant afficher les formules en deux dimensions (sous une forme qui est proche d'une notation sur le papier). Ces logiciels, pour la plupart, utilisaient seulement des caractères alpha-numériques.

Afin d'avoir une idée de ce que l'on peut réaliser avec seulement des caractères alpha-numériques, voici une sortie produite par Macsyma :

```
     /                                n
     [     sqrt(3) t                 ===
2 pi I cos(---------) dt             \      i
     ]         2              t       >    x
     /                       e       /
------------------------ + ------- + ===
         t/2                  2      i = 0
      3 e                f(t,-)
                              3
```

Ces interfaces évoluèrent assez peu durant les années qui suivirent. Ce sont les fonctionnalités mathématiques qui évoluèrent le plus, les systèmes ayant à leur disposition des bibliothèques d'algorithmes de plus en plus importantes et évoluées. Quelques interfaces expérimentales intégrèrent l'utilisation de systèmes de pointage tel que le stylo, pour écran ou tablette, ou la souris.

Il faudra attendre les années 80 et la banalisation du Macintosh et des stations de travail pour trouver des interfaces qui soient plus conviviales. Tous les systèmes actuels présentent une interface graphique qui leur est propre et celle qui paraît la plus avancée est celle de Theorist(89) un logiciel commercial disponible sur Macintosh et sur PC . Depuis, plusieurs publications sur le sujet ont essayé de mettre en évidence les besoins pour la "meilleure" interface utilisateur [Avi90], [Soi91], [Kaj93], [KS94].

Mais la plupart de ces interfaces se contentent de faire la saisie en une dimension et un affichage du résultat en deux dimensions, comme Maple ou Mathematica par exemple et n'utilisent la souris, des menus ou des boutons que marginalement.

Chaque système de calcul formel possède sa propre interface. Afin de s'y retrouver parmi toutes les interfaces existantes, on va d'abord essayer de trouver une classification de celles-ci, puis décrire brièvement les problèmes de saisie et d'affichage des formules.

1.2 Tentative de classification des modèles existants

Après avoir analysé les méthodes de communication entre l'interface et son moteur de calcul, la disposition des requêtes de calculs et des résultats, nous avons pu distinguer trois grandes catégories d'interfaces : le modèle session, le modèle document et le modèle uniforme que nous allons détailler.

1.2.1 Le modèle session

Le premier modèle que l'on peut dégager est un modèle que l'on nommera session ; on pourrait aussi l'appeler classique car c'est celui qui est le plus courant. L'interface est conçue pour gérer un dialogue avec un système de calcul formel sous la forme d'une succession de couples requêtes-réponses qui s'affichent soit dans une même fenêtre, soit dans deux fenêtres séparées. De plus, si l'interface permet l'affichage de graphiques, ceux-ci apparaîssent dans des fenêtres séparées.

Les principaux systèmes utilisant ce modèle sont :

- Maple V avec son interface Iris ([Leo86]).
- Mathematica.
- Macsyma 92.
- Reduce.

Dans la plupart de ces interfaces, la saisie se fait de manière linéaire (en une dimension). L'expression entrée est en fait une expression utilisant un langage de commandes propre au système. La réponse est formatée et imprimée en deux dimensions. Ceci entraîne des difficultés, même des impossibilités à coller les résultats en sortie dans une entrée car à ce moment, on a un mélange d'expressions mathématiques (la sortie), et d'un langage de programmation (l'entrée).

Les interfaces utilisant ce modèle font un placement automatique des formules suivant des règles simples d'affichage. Les questions et les réponses sont positionnées l'une au dessus de l'autre, ou bien les questions sont dans une fenêtre, les réponses dans une autre.

1.2.2 Le modèle document

Avec le modèle document [1], on a des interfaces qui intègrent du texte, des formules et des graphiques qui peuvent être manipulés en vue de former un document imprimable. Maple et Mathematica fournissent aussi ce type d'interface.

Les systèmes utilisant également ce type d'interface sont :

- MathCAD.
- Milo (Intégré dans Framemaker). [Fra93a][Fra93b]
- Scientific Word.

Avec ce type d'interface on a un placement plus intelligent qui dépend du rendu visuel que l'on désire, un peu comme dans un document LaTeX. Ce modèle n'est pas orienté dans le sens interaction "exploratoire", c'est à dire que ce modèle est plus adapté pour la présentation de calculs dans un rapport, avec la possibilité d'évaluer des calculs à la consultation du document électronique.

1.2.3 Modèle uniforme

Le système des questions/réponses que l'on a vu lors de l'étude du modèle session est unifié dans ce modèle ; c'est à dire que l'on n'a pas un format d'entrée et un format de sortie. Toutes les actions que l'on peut effectuer sur la formule saisie sont déclenchés de l'extérieur à l'aide de menus, de *drag and drop* à la souris...

Le principal système utilisant ce type d'interface est :

- Theorist. [BW87]

Ce modèle permet aussi de mélanger les courbes ou graphiques, et les formules, comme le permet le modèle document.

1.3 Les interfaces existantes

Nous allons maintenant nous intéresser plus en détail aux différents points qui posent problème dans les interfaces actuelles comme la saisie, l'affichage, la sélection, la manipulation.

1.3.1 La saisie des formules

Il y a trois méthodes pour la saisie des formules :

- les modèles [2]
- l'*overlay* (du même type que les modèles)

1. aussi appelé notebook ou worksheet
2. *template* en anglais.

- l'analyse syntaxique classique (éventuellement incrémentale)

Traditionnellement, les formules sont saisies de façon linéaire et l'expression ainsi entrée est analysée une fois complète (c'est l'analyse syntaxique classique). L'inconvénient de cette méthode est qu'il est difficile de voir les erreurs syntaxiques (le manque de parenthèses par exemple) mais aussi structurelles (mauvaise expression, oubli d'une sous-expression...) qui sont communes quand on entre des expressions assez compliquées. De plus, cela force l'utilisateur à apprendre et à utiliser une notation qui ne lui est pas forcément familière. On peut remarquer aussi que l'utilisation de cette méthode entraîne un sur-parenthèsage. En fait, on a le même parenthèsage avec une méthode comme celle des modèles que nous allons voir par la suite, mais la plupart d'entre elles sont implicites du fait de la saisie directe en deux dimensions. Comme on peut le voir sur l'exemple suivant, il est beaucoup moins simple de lire la première expression que la deuxième, à cause de cette surcharge des parenthèses. Bien sûr, le formattage en deux dimensions aide à la lisibilité car nous y sommes plus habitués.

- `(1-x^(1/3))/((sqrt((x^2)-1)^(5/4))+1)+x)`
- $\frac{1-x^{\frac{1}{3}}}{\sqrt{(x^2-1)^{\frac{5}{4}}+1+x}}$

Les interfaces modernes (MathScribe [SS86], Theorist [BW87], CAS/PI [Kaj92]) utilisent une forme de saisie bi-dimensionnelle. L'un des principes est l'utilisation de modèles ; on choisit à l'aide d'un menu, d'une palette de boutons ou d'un équivalent clavier le type d'opérateur que l'on désire utiliser. Il en résulte l'affichage de cases à remplir. Par exemple pour une somme, on entre + et il apparait ?+?

Cette méthode présente deux inconvénients majeurs :

- de manière générale, l'utilisation de modèles renforce l'usage de la souris (en particulier pour les novices car ceux-ci ne connaissent pas les raccourcis claviers) ce qui rajoute un coût important pour le temps de la saisie (voir le tableau 1.1).

Opération	Durée de cette opération
Appui d'une touche	$T_K = 0,30s$ [0,08 *à* 1,20]
Pointer une cible avec la souris	$T_P = 1,10s$ [0,8 *à* 1,5]
Passer du clavier à la souris	$T_H = 0,40s$
Préparer mentalement une action	$T_M = 1,35s$

TAB. 1.1 - *Tableau des temps pour les actions élémentaires*

- elle n'autorise pas une saisie naturelle de gauche à droite (comme par exemple pour une intégrale).

1.3.2 Le formatage et l'affichage d'expressions

Les interfaces modernes utilisent toutes des affichages par matrice de points[3] (on ne parlera pas ici des interfaces utilisant seulement des caractères alpha-numériques car celles-ci sont maintenant obsolètes).

Il est très fréquent de manipuler de grosses formules avec des systèmes de calcul formel. En fait c'est une des raisons d'utiliser de tels systèmes. Ainsi, il est très surprenant de s'apercevoir que l'affichage de ces grandes expressions n'est, en général, pas traité différemment par rapport aux petites. Le seul traitement effectué est de couper la formule sur plusieurs lignes. Chaque système produit un effort plus ou moins important pour trouver un endroit approprié pour faire la césure de l'expression. De plus, à cause de ces césures de lignes, des opérateurs ayant un sens verticalement (comme les quotients) perdent une partie de celui-ci.

Certaines interfaces, les plus modernes, fournissent quelques astuces pour résoudre ces problèmes : défilement horizontal, renommage de sous-expressions...

Un autre problème lié à ces grandes expressions est le temps d'affichage. Il y a deux raisons à cela :

- la plupart des systèmes sont basés sur l'utilisation d'arbres pour le formatage. La construction d'un tel arbre est coûteuse : de grandes expressions ont souvent des sous-expressions communes et le calcul pour l'affichage est répété pour chacune d'elles. De plus, en cas de césure, les calculs doivent être refaits et l'algorithme de construction de l'arbre fait des retours arrières qui sont très coûteux.
- le maintien dans un état cohérent de l'arbre qui sert à l'affichage est primordial et peut être d'un coût très élevé si celui-ci n'est pas incrémental, particulièrement en cas de modification d'une formule.

1.3.3 La sélection et l'édition

La plupart des interfaces WYSIWYG[4] intègrent un mécanisme de sélection. Celui-ci peut utiliser un ou plusieurs systèmes de pointage comme la souris, le stylo ... Toutefois, l'utilisation de la souris est le moyen le plus répandu (et même quasiment incontournable pour les novices même si une sélection à l'aide du clavier est parfois bien gérée). Le fait que l'on ait un retour visuel immédiat, par inversion des couleurs, permet de voir simplement si la sous-expression sélectionnée est bien celle désirée.

On peut distinguer deux grands types de sélection :

- la sélection structurelle, qui tient compte uniquement de la syntaxe sous-jacente.

3. *bitmap display*
4. WYSIWYG : "What you see is what you get"

– la sélection non structurelle, uniquement basée sur la géométrie de l'image affichée.

De plus, ces deux types de sélection peuvent être combinées avec la notion de sélection multiple (plusieurs “entités” sélectionnées en même temps).

Mais, souvent, les interfaces ne contrôlent pas la sémantique lors de la sélection, considérant l'expression comme une simple chaîne de caractères. Par exemple dans Maple, on peut sélectionner une sous-expression qui n'est pas forcément valable syntaxiquement

De plus, les sélections ne correspondant pas à la représentation interne sont souvent problématiques voir même impossibles (la diagonale d'une matrice par exemple) et la sélection multiple de sous-expressions est inexistante.

1.3.4 La manipulation des formules

Un exemple d'opération élementaire difficile à réaliser dans les interfaces des sytèmes les plus courants, est l'expansion d'une partie d'une formule.

Par exemple pour appliquer `expand` seulement sur le deuxième terme d'une somme obtenue en résultat d'une opération précédente, il faut avec Maple donner une commande comme `op(1,") + expand(op(2,"))` (si toutefois la somme n'a que deux termes). Il est évident que l'on pourrait, à l'aide des mécanismes de sélection, réaliser cette opération d'une manière bien plus simple.

Une autre possibilité offerte (mais trop rarement) est une manipulation sémantique des expressions mathématiques. L'un des premiers logiciels a avoir utilisé cette technique est Milo qui a été développé pour aider à la résolution de problèmes élémentaires de physique. Depuis, Milo a été intégrée dans le traitement de texte FrameMaker ([Fra93a], [Fra93b]). La figure 1.1 est un exemple illustrant cette technique. L'élément x est sélectionné et on obtient les équations suivantes par une opération de déplacement à droite.

$$\boxed{\frac{\boxed{x}\times y}{2}+x=1 \quad \frac{y\times\boxed{x}}{2}+x=1 \quad \frac{y}{2}\times\boxed{x}+x=1 \quad (\frac{y}{2}+1)\times\boxed{x}=1 \quad \frac{y}{2}+1=\frac{1}{\boxed{x}}}$$

FIG. 1.1 - *Exemple d'application utilisant la méthode de Milo*

Mais ces possibilités sont encore trop souvent absentes ou pas assez exploitées au sein des interfaces actuelles.

1.4 Les problèmes

Les principaux problèmes des interfaces actuelles sont liés aux modes de saisie des formules, à la manipulation de celles-ci et à l'organisation des calculs.

Nous avons étudié les différents points techniques qui posent problème dans une interface pour le calcul formel, à savoir :

- le mode de saisie des formules qui utilise la souris en alternance avec le clavier. Cette combinaison n'est pas optimum car on perd beaucoup de temps lors de la saisie. Il vaudrait mieux tout faire au clavier ou à la souris. Le tout souris est difficilement envisageable, donc nous nous orienterons sur une méthode de saisie utilisant au maximum du clavier.
- l'affichage qui ne fait pas la différence entre des grandes formules, des grandes matrices, ou de simples petites formules. Il faudrait donc palier à cette lacune en introduisant des mécanismes nouveaux (autre que la césure). Par exemple, on pourrait envisager de faire une représentation partielle d'un grand polynôme, en n'affichant que les termes de plus haut et de plus bas degré. Malheureusement, dans la plupart des interfaces, trouver une bonne "représentation partielle" réclame du calcul et la mise en œuvre d'algorithmes particuliers.
- les mécanismes de sélection qui à mon sens n'utilisent pas assez le système de sélection multiple.
- la manipulation des formules et surtout des grosses formules. Cette manipulation peut-être facilitée en faisant un usage plus important de la sélection multiple.

Nous allons maintenant tenter de résoudre ces problèmes en proposant la spécification d'un nouveau modèle d'interface pour le calcul formel.

Chapitre 2

Un autre modèle d'interface pour le calcul formel

Notre but est d'avoir un modèle d'interface avec lequel un utilisateur (ingénieur, mathématicien) retrouve un confort proche de celui qu'il a avec sa feuille de papier et son crayon (ou son tableau). Cela signifie :

- saisie “intuitive” et impression en deux dimensions.
- placement libre des formules et des résultats de calculs.
- manipulation des expressions à la souris et à l'aide de menus.
- possibilité de suivre plusieurs calculs possibles.

Les modèles actuels des interfaces pour le calcul formel sont assez intéressants. Chacun présente un point de vue différent, sur la manipulation, la présentation des calculs, et ceci peut être intéressant suivant ce que l'on désire faire (soit des calculs, ou une présentation de formules s'intégrant dans un document). Toutefois, ces modèles sont assez éloignés du comportement naturel que l'ont peut avoir devant une feuille de papier.

Ceci nous conduit donc à introduire un autre modèle d'interface pour l'édition et la manipulation de formules mathématiques.

2.1 Un nouveau modèle : le tableau noir

L'objectif est de pouvoir placer et manipuler des formules comme sur un tableau ou sur un feuille de papier, comme on a l'habitude de le faire.

Pour pouvoir faire cela, nous devront fournir des mécanismes qui soient les plus proches possibles de ce que l'utilisateur à l'habitude de faire sur une feuille de papier avec un crayon, avec comme avantage supplémentaire d'exploiter tout ce que la machine peut apporter.

- Saisie naturelle qui soit la plus proche possible de ce que l'on peut faire sur une feuille, avec des mécanismes de repositionnement automatique.
- Manipulations des formules agréable grâce à la multi-sélection et au *drag and drop* réalisé à l'aide de la souris ; avec des opérations possibles comme la simplification, l'évaluation, la factorisation.

Mais certaines de ces caractéristiques, comme la saisie naturelle, sont assez subjectives et dépendent de l'utilisateur. L'interface bâtie à l'aide de ce modèle doit pouvoir s'adapter facilement aux désirs de l'utilisateur. Il faut donc avoir accès un certain degré de paramétrisation. Ainsi, cette approche nous permettrait de construire un prototype et d'effectuer des tests à court terme, pour pouvoir déterminer des comportements qui paraîssent les plus prometteurs.

Ce modèle est à mi-chemin entre le modèle session et le document car il doit permettre de positionner les formules où l'on veut mais n'est pas destiné à en faire un document imprimable, et est plus orienté sur le point de vue des calculs.

Notre modèle a une approche différente du modèle session (sur le fond). Dans le modèle session, on exécute des commandes du système (`expand, sqrt, sin...` sont des commandes que le système interprète). Avec le nouveau modèle, on se contente de saisir et de manipuler des formules puis, on va chercher les commandes concernant tout ce qui est calcul, ailleurs grâce à la connexion à un système de calcul. Cela permet d'être indépendant du système auquel on est connecté.

2.2 Organisation des calculs

L'organisation des calculs est une partie primordiale de l'interface. Il faut donc que ceci soit facilement paramétrable pour pouvoir adapter l'interface selon son désir et ses besoins.

Nous allons spécifier quelques comportement qui paraîssent intéressants pour l'organisation de sa feuille de calcul.

- Le positionnement des formules sur la feuille de calcul.
- L'organisation d'un ensemble de formules sur des *tableaux*.
- La sauvegarde d'une session.

2.2.1 La disposition des formules

Le but principal est de fournir à l'utilisateur la plus grande liberté pour le positionnement de ses formules comme il peut le faire sur une feuille de papier. Il ne doit pas être limité à une suite trop linéaire de calculs comme dans la plupart des interfaces actuelles.

Lors de la manipulation des formules, et plus particulièrement quand le système de calcul renvoie un résultat, il faut déterminer si ce résultat va remplacer la formule actuelle (celle qui est à l'origine du calcul), ou bien si c'est une nouvelle formule, à part entière.

On doit donc avoir un mécanisme permettant à l'utilisateur de dire ce qu'il veut faire, à savoir les modifications en place ou créer une nouvelle formule.

2.2.2 L'organisation en *tableau*

En ce qui concerne l'organisation d'un groupe de formules, nous pensons fournir deux mécanismes.

L'utilisateur doit pouvoir mettre de côté un ensemble de calculs afin de poursuivre dans une autre direction. Il aura alors la possibilité d'ouvrir un nouveau tableau où il poursuivra ses calculs à partir de formules qu'il reprendra sur un autre tableau ou bien de nouvelle(s) formule(s).

L'utilisateur pourra aussi tout simplement masquer tout ou partie de ses formules pour avoir un "plan de travail" plus clair sans perdre les calculs qu'il a déjà effectué. Le fait de masquer les formules créera un tableau virtuel où les formules seront toujours présentes. Ces formules seront alors remplacées par une marque les identifiant (icône ou vue réduite). Cette marque symbolisera le tableau virtuel et on devra pouvoir le nommer afin que l'utilisateur puisse l'identifier facilement. Une autre solution est de créer une image réduite de l'ensemble des formules que l'on fait disparaître. Mais cette solution reste à tester car il semble difficile de différencier deux paquets de formules sur une vue réduite.

L'utilisateur désirant retrouver ses calculs pourra alors soit ré-expanser les formules sur le même tableau ou bien dans un nouveau tableau ce qui correspond à rendre le tableau virtuel visible.

2.2.3 Sauvegarde d'une session

Deux possibilités s'offrent à nous en ce qui concerne la sauvegarde d'une session. Soit on effectue une sauvegarde qui tient compte de l'emplacement et des attributs visuels des formules, soit une sauvegarde simple des calculs.

Les deux points de vue sont intéressants. La première solution permet de retrouver le travail dans l'état où on l'a laissé. La seconde solution quant à elle, privilégie les formules sur le rendu visuel. On considère alors la sauvegarde comme un archivage des calculs intéressants. Des tests auprès des utilisateurs devraient permettre de déterminer quelle solution adopter.

2.3 Manipulation des formules

Le deuxième objectif du modèle d'interface est de pouvoir manipuler de façon intuitive les formules pour les opérations classiques. Pour s'abstraire des commandes dépendant du système de calcul et d'un apprentissage de commandes, on voudrait effectuer le maximum d'opérations à la souris. Un complément pourra être fourni à l'aide de menus ou de boutons. Le fait de vouloir faire des manipulations de formules à la souris n'est pas contradictoire avec le fait que l'on veuille s'abstenir au maximum de l'utilisation de la souris lors de la saisie. C'est le va et vient clavier/souris qui est pénalisant et peu ergonomique. Il n'est pas non plus désirable d'avoir une main sur chacun des périphériques car la saisie de formules implique des combinaisons de touches (shift $+$, shift $)$, ...) qui sont difficilement réalisables avec une seule main.

Donc la manipulation quasiment entièrement à la souris paraît une bonne idée. Nous allons voir les différents types de manipulation possibles avec les mécanismes existants : sélection multiple, *drag and drop*.

2.3.1 La sélection multiple

Nous pouvons modéliser les intéractions qui peuvent être effectuées à l'aide de la sélection multiple suivant le prototype de la fonction que l'on veut appliquer. L'idée est d'utiliser plusieurs sélections pour indiquer les différents arguments d'une opération et ensuite d'utiliser un menu, un bouton ou une séqunce de touches pour lancer l'opération. Cette possibilité peut être très agréable en pratique (cf l'exemple de l'expansion d'une sous-expression dans le paragraphe 1.3.4).

Nous avons choisi un ensemble de fonctions classiques et nous verrons que l'on peut donner un modèle d'intéraction suivant ce prototype.

- Expansion : `Expr ->Expr` (Expand).
- Factorisation 1 : `Expr ->Expr` (Factor).
- Factorisation 2 : `Expr * List(Var) ->Expr`.
- Resolution : comme factorisation peut être de deux types.
- Substitution : `Expr * (Var, Expr) ->Expr` (Substitute).
- Simplification : `Expr ->Expr(s)` (Simplify).

Pour les fonctions du type `Expr ->Expr` :
Le mécanisme est : sélection de l'expression sur laquelle on travaille.

Pour les fonctions du type `Expr * List(Var) ->Expr` :
Sélection de l'expression sur laquelle le traitement doit s'effectuer, puis sélection de n variables (avec test du type de ce qui est sélectionné) ; donc le mécanisme de multi-sélection qui entre en jeu.

Pour les fonctions du type `Expr * (Var, Expr) ->Expr` :
On a aussi un mécanisme de multi-sélection pour l'expression, la variable et l'expression qui remplace la variable (pas forcément dans la même formule).

2.3.2 *Le Drag and Drop*

Une autre technique peut être envisagée pour réaliser une opération comme la substitution. A l'aide de la technique du *drag and drop*, on peut réaliser de manière intuitive cette opération. Après avoir sélectionné une sous-expression, on peut la prendre et la déposer sur une variable ce qui signifierait de remplacer la sous-expression par la variable au sein de la formule.

Suivant une technique similaire, il est possible (comme dans Theorist) de réaliser des résolutions d'équations (passage d'un membre à l'autre). Néanmoins cette possibilité ne semble pas encore s'inscrire dans notre modèle du fait qu'il faut connaître le "type" de l'équation. Mais, des tests permettront de voir si cette solution est viable.

2.3.3 En intéraction avec une base de formules

D'autres opérations sont envisageables en intéraction avec une base de formules. Ce composant, en court d'étude dans le projet SAFIR, est une base de données déductive d'identités mathématiques interrogeable par un langage de requête spécialisé. Chaque opérateur pourrait avoir un menu associé contenant des équivalences à cet opérateur. Par exemple pour un opérateur comme `sin(x)`, le menu contiendrait des formes equivalentes comme $\cos(\frac{\pi}{2} - x)$, $sin(\pi - x)$, $-cos(x + \frac{\pi}{2})$, ... L'utilisateur aurait alors le loisir de choisir une forme équivalente pour faire la substitution.

Chapitre 3

Présentation d'Emath

3.1 Le serveur EMATH

Une architecture du type client-serveur a été adoptée pour pouvoir bien faire la séparation entre un composant fournissant les primitives d'édition de formules mathématiques (EMATH) d'une part et l'interface graphique et la manipulation des formules d'autre part. Cet éditeur est écrit en C mais il comprend aussi une partie Klone (un dialecte de Lisp, très proche de Common Lisp) qui est utilisé comme langage d'extension (un peu à la manière de l'éditeur de texte Emacs qui utilise Emacs-Lisp). Klone a été conçu pour être embarqué dans des applications et pour être facilement extensible.

Actuellement, EMATH est un composant qui représente approximativement 14000 lignes de code C et 1500 lignes de Lisp, pour le noyau minimal.

EMATH est un composant qui est développé par Olivier Arsac dans le cadre de sa thèse. J'ai travaillé en étroite collaboration avec lui.

3.2 L'intégration de EMATH dans une interface

Une fois le serveur EMATH lancé, l'application peut communiquer avec lui en envoyant des s-expressions Lisp. L'interface peut déléguer au serveur une ou plusieurs fenêtres pour afficher et éditer des formules. Ce mécanisme de délégation peut intervenir à tout moment (pas seulement au démarrage du serveur), juste en exécutant la fonction Lisp `manage-window` qui prend pour argument l'identificateur X de la fenêtre.

```
(setq current-window (manage-window ...))
```

Ainsi, l'application délègue une formule à EMATH et celui-ci va s'occuper de tout ce qui concerne l'affichage de la formule.

```
(setq test-formula '(/ (+ a b) c))
(display-formula test-formula current-window)
```

De manière pratique, le programmeur qui a besoin d'éditer et d'afficher des formules doit effectuer les trois étapes suivantes :

- Créer une hiérarchie de widgets (pouvant inclure des barres de menu, des boutons...) avec un ou plusieurs widget(s) neutre(s) (comme les widgets de dessin [1]). Puis, on délègue ces widgets à EMATH grâce à la fonction `manage-window` vue précédement. EMATH se chargera alors de l'affichage des formules que le client lui envoie à l'aide de la fonction `display-formula`.
- si nécessaire, écrire des fonctions Lisp pour paramétrer EMATH comme le veut l'utilisateur. Une bibliothèque minimale des comportements "standards" d'édition et d'affichage des formules est fournie (cf chapitre 4).
- Ecrire des fonctions de traduction de la représentation utilisée par EMATH et de celle utilisée dans l'application.

3.3 L'affichage de formules

Pour afficher une formule sur l'écran, trois étapes principales sont nécessaires :

- Décider du format à utiliser pour représenter chaque opérateur (décider qu'un quotient est représenté avec une barre de fraction horizontale ou oblique, qu'un exposant est dans une police plus petite et sur la droite de la variable auquel il se rattache...).
- Calculer les images de chaque caractère intervenant dans la représentation de la formule. Normalement, pour les opérateurs les plus simples, pour les variables et les nombres, on se contente de trouver le caractère correspondant dans la fonte adéquate. Mais pour des opérateurs comme la somme ou l'intégrale qui doivent être re-dimensionnable, plus de travail est nécessaire.
- afficher toutes ces images à la bonne place sur la fenêtre.

Le processus de formatage peut être complètement ou partiellement sous le controle de l'interface. Ceci permet d'implémenter de manière assez simple plusieurs types d'interfaces correspondant aux modèles existants (cf chapitre 2 sur les modèles).

3.4 L'architecture interne d'EMATH

3.4.1 Les arbres attribués

La structure généralement utilisée pour la représentation et l'affichage de formules mathématiques est l'arbre. EMATH ne fait pas exception à la règle. Cette structure est utilisée pour contenir la formule en elle même mais aussi les attributs utilisés notamment pour l'affichage.

Chaque nœud d'une formule peut porter des attributs. Les attributs sont utilisés par EMATH pour spécifier des paramètres d'affichage (comme la couleur, la police de caractères

1. drawing area

...). Les attributs peuvent être consultés et modifiés par le client et celui-ci peut aussi ajouter ses propres attributs.

Les attributs ne sont pas répétés à chaque nœud de l'arbre car ceci induirait un coût trop important de stockage. Un système d'héritage permet d'éviter ce problème. Dans un premier temps, on a un héritage au niveau de l'arbre d'une formule. Les attributs qui sont valables pour toute la formule sont stockés à la racine de l'arbre. Bien sûr, on peut redéfinir localement ces valeurs, d'où la notion d'héritage. Quand on cherche un attribut, on regarde tout d'abord si celui-ci est présent au niveau du nœud, si non, on remonte au père et ainsi de suite jusqu'à la racine. En fait, un mécanisme interne, transparent pour l'utilisateur, permet d'accéder plus rapidement à la ressource voulue. Cette optimisation était nécessaire car on consacre beaucoup de temps à la recherche de la valeur des attributs.

EMATH est un serveur qui se veut multi-application, multi-écran et multi-fenêtre. Donc le mécanisme d'héritage de ces attributs doit être valable aussi pour ces caractéristiques. C'est pour cela qu'un structure, que l'on appelle l'arbre des groupes, a été mise en place. C'est aussi une structure arborescente qui regroupe dans ces différents nœuds les arbres des formules et on peut positionner des attributs sur les nœuds de cet arbre. Ainsi, des attributs sont valables pour une fenêtre, un écran (au sens X du “display”), une application ou pour toute application utilisant EMATH. Cette organisation a été inspirée du mécanisme des ressources X.

Lorsqu'on lance EMATH, l'arbre se réduit à la racine seulement. A chaque connection d'une application au serveur, on crée un nœud pour cette application et un nœud suivant l'écran sur lequel s'affiche l'application. Puis, pour chaque fenêtre que l'application délègue à EMATH, on crée un nœud et on y attachera la formule que l'application donne à gérer à EMATH.

Cette notion d'arbre de groupe est illustrée par la figure 3.1.

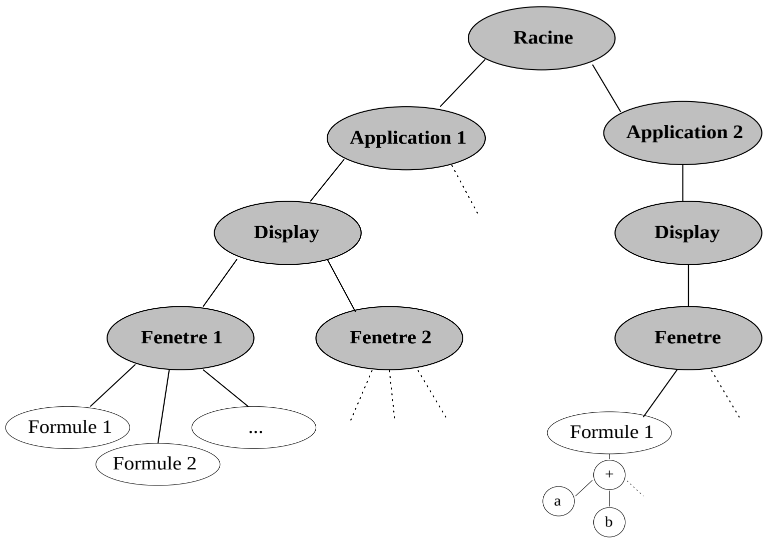

FIG. 3.1 - *L'arbre des groupes et des formules*

Pour créer un nouvel attribut, il suffit de l'ajouter dans la liste des attributs connus par EMATH. Cet ajout peut se faire au niveau des ressources de l'interface ou bien directement dans le code. Voici un exemple de déclaration des attributs par une ressource :

```
Application.*.list-of-attributes:(
   (foreground Pixel)
   (background Pixel)
   (body-font-0 FontStruct)
   (body-font-1 FontStruct)
   (body-font-2 FontStruct)
   ...
   (body-font-smallest FontStruct)
   (body-font-color Pixel)
   (show-impl-mult Boolean)
)
```

Pour affecter une valeur à un attribut, deux méthodes sont possibles :

- par le mécanisme des ressources X :

  `*.body-font-color: White`

  `*.show-impl-mult: False`
- directement dans le code Lisp.

Pour affecter en Lisp une valeur à un attribut on fait appel à la fonction `aset` (set attribut). Les paramètres à fournir sont : le groupe et le nœud sur lequel on veut positionner cet attribut, l'attribut concerné et sa valeur :

```
(aset group node "background" 'DarkGray)
```

Pour connaître la valeur d'un attribut, il faut utiliser `aget` qui possède les mêmes caractéristiques que `aset` mis à part la valeur à donner évidemment.

```
(aget group node "background") -> DarkGray
```

Si l'on désire qu'un attribut soit visible pour tous les nœuds de l'arbre, il faut positionner cet attribut sur le nœud racine de l'arbre. Pour éviter d'avoir à rechercher systématiquement le nœud racine pour le passer en paramètre, une petite "optimisation" est fournie ; il suffit de passer liste vide (). Par exemple, si on veut positionner un attribut qui soit visible pour toutes les applications (comme c'est le cas pour les propriétés de base des opérateurs d'EMATH), il suffit de faire :

```
(aset () () "op-properties" (parse-properties))
```

Les `aset` et `aget` peuvent être valables seulement sur le nœud sur lequel on fait la requête. Pour cela, on a les fonctions `aset-local` et `aget-local` qui sont sur le même type que les fonctions générales correspondantes.

Enfin, une fonction permet de supprimer un attribut qui a été positionné : `aunset` et la fonction locale correspondante `aunset-local`.

3.4.2 Serveur multi-processus

La construction d'un arbre, le calcul des boîtes englobantes et l'affichage sont autant d'opérations qui prennent du temps machine, même en utilisant une méthode incrémentale. Classiquement, on traite un événement (comme l'appui sur une touche) et on fait un réaffichage si nécessaire. Mais cette technique implique une perte de réactivité de l'application. En effet, durant le temps où on traite un événement, les autres sont bloqués, dont les événements concernant le réaffichage. Il a donc été décidé de détacher l'événement de rafraîchissement écran des autres. Dans EMATH, les événements sont traités suivant leur priorité. L'affichage ne se produit que quand il n'y a rien d'autre à faire et est interrompu si un autre événement arrive. Des processus légers[2] sont utilisés pour effectuer cette tâche.

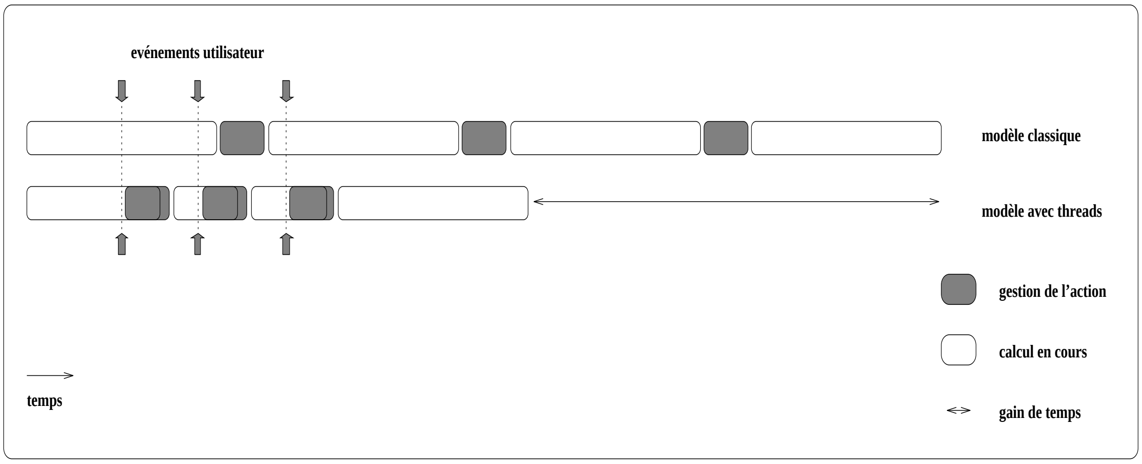

FIG. 3.2 - *Comparaison des méthodes de rafraîchissement*

La figure 3.2 permet de montrer que la méthode utilisant les processus légers se termine plus rapidement que la méthode utilisant un modèle classique de rafraîchissement. Sur ce schéma, le temps s'écoule de gauche à droite. Les bandes blanches symbolisent le temps de calcul nécessaire avant que le rafraîchissement ne soit valide, et les bandes grisées, le temps consacré au traitement de l'événement. Les flèches épaisses symbolisent le moment où un événement impliquant un rafraîchissement arrive.

Avec le modèle classique, les événements provoquant un rafraîchissement, ne sont traités que lorsque le calcul du rafraîchissement en cours est terminé.

Avec le modèle des threads, dès que l'événement arrive, on interrompt le calcul en cours. Il est possible de faire cela car si un événement de rafraîchissement arrive c'est que le calcul qui était en cours est obsolète car de nouvelles données vont modifier le résultat. Donc on interrompt le calcul afin de recommencer au plus vite un calcul qui soit valide. On voit très bien sur le schéma, le temps que l'on gagne à l'aide de cette méthode.

2. threads

3.4.3 Utilisation mémoire

Pour chaque formule, il existe un grand nombre d'attributs pour pouvoir spécifier les propriétés graphiques. Certains attributs sont prédéfinis et l'utilisateur peut en ajouter des nouveaux. Un mécanisme d'héritage est utilisé pour factoriser au maximum les instances de ces attributs. Ainsi, si tous les nœuds d'un arbre partagent la même valeur pour un attribut, celui-ci n'est stocké qu'à la racine de l'arbre (cf 3.4.1).

Donc, la recherche de la valeur pour un attribut peut engendrer un parcours de l'arbre, sur sa hauteur au maximum.

De plus, seule la structure relative à l'affichage est maintenue pour la partie visible de la formule. Ainsi, la mémoire utilisée pour l'affichage ne dépend pas de la taille de la formule mais de la taille des fenêtres contenant des formules. La taille pour un nœud de formule est de 15 octets, dans l'implémentation actuelle.

Chapitre 4

Une bibliothèque pour l'interface

Nous allons reprendre les différents problèmes vus dans le chapitre 1 et nous expliquerons les solutions retenues. Ces solutions ont été, pour partie, implémentées et forment une bibliothèque minimale des comportements souhaitables pour l'édition, la sélection...

4.1 La saisie des formules

Comme nous avons pu voir dans le paragraphe 1.3.1, la saisie de l'expression est une étape fondamentale. Mais chacune des solutions existantes présente des désavantages importants. Nous nous sommes donc penchés tout d'abord sur ce problème.

On se donne pour objectif d'avoir une méthodologie qui soit conviviale pour tous les utilisateurs, du novice à l'expert. De plus, le système doit être entièrement paramétrable car on doit concevoir un outil de recherche donc facilement adaptable aux besoins.

Nous avons orienté nos recherches suivant deux directions : une première méthode utilisant comme principe de base un automate à états, la deuxième méthode consisterait en une spécification par motifs d'arbres.

Dans un premier temps, nous optons pour la méthode de l'automate que nous tenterons d'exploiter totalement. Si cette méthode s'avère insuffisante, nous combinerons celle-ci avec les motifs d'arbres.

Après étude, nous nous sommes rendus compte que tout ce qui pouvait être exprimé grâce aux motifs d'arbres pouvait aussi l'être avec la méthode que l'on a retenue. Mais en plus, la méthode utilisant les motifs d'arbres seule (sans notion d'état) ne permet pas d'exprimer la notion d'historique. Pour expliciter cette notion, nous allons nous appuyer sur l'exemple illustré par la figure 4.1.

$$a^2 \qquad\qquad a^2 + b$$

FIG. 4.1 - *Exemple*

Dans la partie de gauche, on a une saisie qui se fait en continu. L'utilisateur frappe au clavier a ˆ 2 (a^2) et l'on veut que le comportement "standard" soit que s'il tape un opérateur (par exemple le plus), on revienne automatiquement à la ligne pour continuer la saisie. C'est un comportement souhaitable lors de la saisie de polynômes. Par contre, comme l'illustre la deuxième partie de la figure 4.1, si on vient faire une modification d'une formule déjà entrée dans le système, le comportement doit être différent. En effet, si celui-ci est toujours identique, on ne pourra jamais avoir quelque chose du type $a^{2+n} + b$. Mais au niveau de l'arbre qui représente l'expression, on ne peut pas faire la différence entre la saisie en continu et une rectification dans une formule existante ; l'arbre est le même.

Le mécanisme utilisé permettant cette plus forte expressivité est un automate. Pour mieux voir comment celui-ci est construit, nous proposons le schéma 4.2 qui est un automate de saisie typique, optimisé pour des polynômes. Nous ne ferons que quelques remarques sur ce schéma. La différence majeure par rapport à un automate classique est que l'événement qui permet de passer d'un état à un autre n'est pas forcément "consommé" ; seules les transitions en pointillés absorbent l'événement qui a provoqué la transition. L'état Mult-Impl est utilisé pour la notion de multiplication implicite.

La séquence suivante permet de traverser les états NoState, Token, Number, Mult-Impl : *2 a*

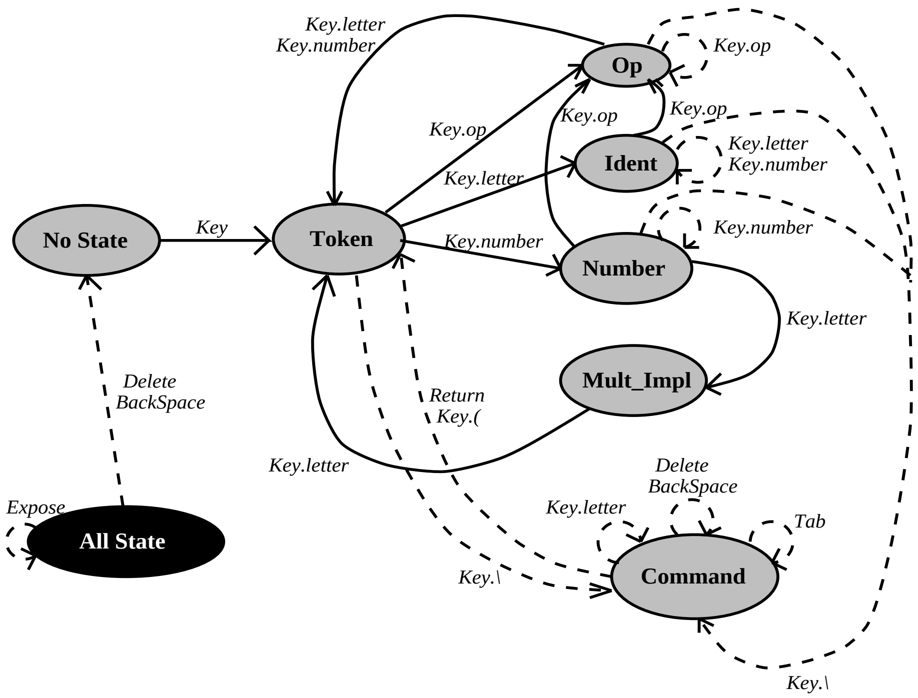

FIG. 4.2 - *Exemple d'automate optimisé pour la saisie de polynômes*

Pour l'implémentation de cette méthode, on utilise une extension des tables de traduction de la boîte à outils (toolkit) X. Cette extension des tables de traduction définit quelle action doit être exécutée quand un événement est détecté, ceci respectant un contexte d'exécution. Voici un exemple typique d'extension des tables de translation qui permet d'implémenter la notion d'automate que nous avons décrit :

state	event	action	next state
`selection-activated`	<Key>LeftArrow	`move-selection-left`	
`selection-activated`	<Key>Space	`extend-selection`	
`selection-activated`	<Key>[a-zA-Z]	`select-to-edit`	`edition`
`edition`	<Key>LeftArrow	`move-cursor-left`	
`edition`	<Key>Delete	`delete-current-node`	

L'appui sur la touche flèche droite déclenche une action qui dépend de l'état dans lequel on se trouve (soit `selection-activated`, soit `edition`). Pour ces deux événements, on ne change pas d'état dans l'automate.

Après avoir implémenté l'automate et les fonctions de création et de destruction d'une formule, nous avons pu nous apercevoir que les fonctions écrites sont toutes assez complexes. Pour chacune d'elle, il faut faire des tests afin de connaître le type du père, des frères ou même du grand-père. Ces opérations répétitives nous ont conduit à penser qu'il faudrait une technique pour simplifier la réutilisabilité et la lisibilité des fonctions (ce qui les rend plus facilement adaptables).

La solution que nous envisageons pour résoudre ce problème est d'écrire un petit langage sur les motifs d'arbre. Ceci nous permettrait de simplifier les fonctions. Donc le meilleur compromis serait probablement un mélange des deux méthodes.

Un exemple détaillé de code est fourni en annexe 1.

4.2 Défaire les opérations effectuées

La méthode qui consiste à pouvoir défaire les opérations dans l'ordre inverse où on les a effectuées (comme dans l'éditeur de texte emacs) est trop souvent absente des interfaces pour le calcul formel. Le seul outil proposé allant dans ce sens est une possibilité de défaire la dernière opération effectuée.

Grâce à la méthode de l'automate, il est relativement simple de mettre en place cette fonctionnalité. Il suffit de mémoriser dans une liste les transitions effectuées dans l'automate. Ainsi, pour défaire une succession d'opérations, il suffit d'appeler les fonctions *"inverses"* de celles qui ont été stockées dans la liste.

4.3 La sélection

Une première bibliothèque de fonctions a été mise au point pour avoir un mécanisme de sélection. Ce mécanisme a été un peu complexe à mettre en place du fait de l'architecture un peu spéciale de l'application totale (EMATH et l'interface). En effet, il faut que la sélection soit visible à la fois du côté du serveur EMATH et de l'interface.

La sélection peut être multiple (cf chapitre 2). Par exemple, en sélectionnant plusieurs sous-expressions, on peut les combiner suivant un opérateur : on sélectionne plusieurs sous-expressions syntaxiquement correctes et en appuyant sur $+$, on pourra recombiner les différentes sous-expressions en une nouvelle qui sera la somme des sous composants ; ou encore demander la résolution d'une expression sélectionnée suivant une variable...

4.4 Les déplacements

Les déplacements dans les formules sont souvent critiquables pour les différentes interfaces testées. En effet, lorsque l'on se déplace à l'aide des flèches dans une formule, ce déplacement se fait de manière syntaxique et non spatiale comme l'utilisateur peut s'y attendre. La figure 4.3 est une illustration de ce que l'on vient d'expliquer. Si le curseur se trouve sur l'identifiant a et que l'on fait une action qui correspond à vouloir se déplacer vers le bas (flèche vers le bas par exemple), on désirerait que le curseur aille sur c et non pas simplement remonter dans l'arbre syntaxique, ce qui correspond à une sélection de $a + b$.

Pour remédier à cela, on élabore un algorithme permettant de déterminer la boîte englobante la plus proche spatialement et suivant une direction, de la boîte où l'on se trouve. Ceci permet d'avoir un comportement un peu dans le style de ce qui est disponible sous emacs. Par exemple, quand le curseur est en fin de ligne et que l'on passe à la suivante, le curseur est toujours positionné le plus près possible de la position verticale qu'il avait avant le déplacement.

Syntaxique:

$$\frac{\boxed{a}+b}{c} \longrightarrow \frac{\boxed{a+b}}{c} \longrightarrow \boxed{\frac{a+b}{c}}$$

Spatial :

$$\frac{\boxed{a}+b}{c} \longrightarrow \frac{a+b}{\boxed{c}}$$

FIG. 4.3 - *Comparaison déplacement spatial et syntaxique*

4.5 La traduction en LaTeX

Nous nous sommes rendu compte que certains utilisateurs avaient recours à l'interface de Maple afin de saisir leurs formules pour les exporter au format LaTeX. Cette démarche, qui peut paraître étrange, s'explique par le fait qu'il n'existe rien de moins coûteux que de démarrer un système de calcul formel pour pouvoir visualiser facilement des formules mathématiques. Nous avons donc pensé qu'il pouvait être intéressant de proposer ce mécanisme de traduction en LaTeX dans notre bibliothèque.

La partie de la formule qui est sélectionnée peut être exportée dans une autre application, comme un éditeur de texte, sous sa forme LaTeX.

Pour faire cette traduction en LaTeX de la forme arborescente de la formule, on utilise une structure contenant la correspondance entre un opérateur de EMATH et une fonction permettant d'ajouter l'opérateur sous sa forme de commande LaTeX dans la chaîne de caractères résultat. Le parcours de l'arbre effectué est une descente en profondeur d'abord. Les opérateurs ayant un comportement similaire, à la fois en ce qui concerne leur structure et leur affichage, sont regroupés en classes, ce qui permet d'éviter de dupliquer le code sur ces opérateurs qui sont "similaires". Des opérateurs comme sin, cos... sont regroupés, mais aussi comme $+$ et $*$ (opérateurs n-aires).

Des attributs positionnés sur les nœuds de l'arbre qui contient la formule, permettent de spécifier l'aspect que doivent prendre les opérateurs. Par exemple, pour la fraction, on doit avoir une information qui dit si on représente l'opérateur comme une fraction ($\frac{a+b}{c}$) ou avec une barre oblique ($(a+b)/c$).

4.6 L'implémentation

Les différentes méthodes que l'on a pu voir au cours de ce chapitre ont été implémentées et forment une bibliothèque qui vient s'intégrer dans le client que l'on décrit dans le chapitre 5. Deux langages ont été utilisés pour l'implémentation des fonctions :

- Klone : un Lisp qui permet d'être embarqué facilement dans une application et d'appeler des fonctions écrites en C. Tout ce qui concerne l'automate et les fonctions associées, la traduction en LaTeX et les déplacements ont été implémenté grâce à ce langage.
- C : les fonctions sur les mécanismes de sélection ont été écrites dans ce langage.

Cette bibliothèque représente environ 3000 lignes de Lisp et 500 lignes de code C.

Chapitre 5

Une interface

5.1 L'interface cliente

L'interface graphique que l'on a construit vient se connecter sur le serveur EMATH. Cette interface a été baptisée M&M (Maple et Mathematica) car elle est indépendante du système de calcul et devrait permettre d'accéder aussi bien à Maple que à Mathematica. On retrouve sur le schéma 5.1 tous les éléments que l'on a présenté, à savoir : EMATH, la bibilothèque des fonctionnalités d'édition et de manipulation, et la partie interface que l'on va présenter maintenant.

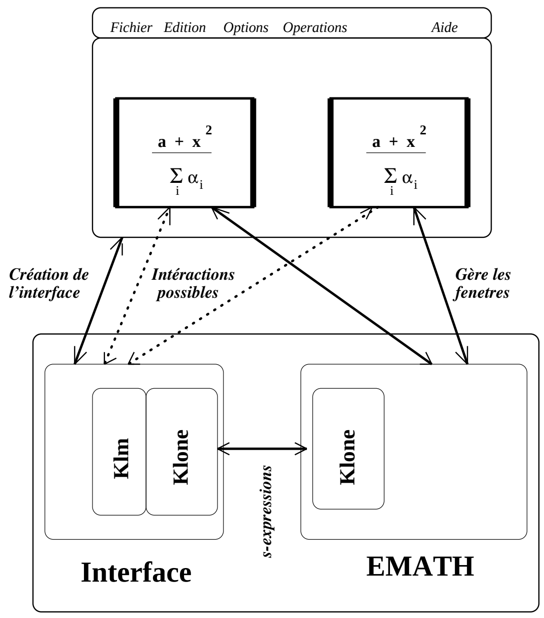

FIG. 5.1 - *La connexion Interface*-EMATH

Cette interface est écrite en Klm (Klone-Motif). Klm est une bibliothèque permettant d'appeler des fonctions Motif en Lisp (Klone). Notre choix s'est porté sur cette bibliothèque car la quantité de code à fournir pour construire les éléments de base d'une interface est bien moindre que si le travail était effectué en C. De plus, on évite les compilations et cette sur-couche à Motif a été construite de telle sorte que certains mécanismes répétitifs soient cachés. Le code intéressant n'est donc pas dispersé au milieu de dizaines de lignes sans intérêt. Pour plus de détails sur Klm et sur le code qui compose l'interface voir l'annexe 2.

L'interface peut être connectée à un ou plusieurs systèmes de calcul formel par l'intermédiaire du CENTRAL CONTROL [DGS94], [DGS95]. Ce composant (développé au sein du projet SAFIR) peut-être vu comme une boîte noire qui permet de se connecter de manière transparente à plusieurs systèmes de calcul formel.

Pour mieux comprendre l'architecture complète et les interconnexions entre EMATH, l'interface et le(s) système(s) de calcul formel, voici le schéma 5.2 qui présente toutes ces informations.

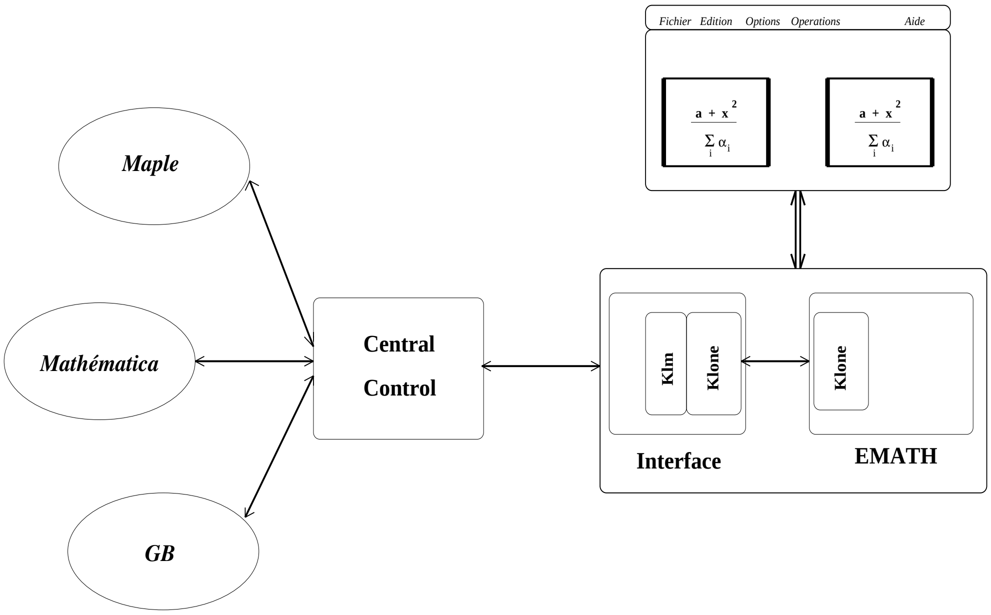

FIG. 5.2 - *La connexion Interface-Central Control*

5.2 L'interface M&M

Pour pouvoir tester différents mécanismes de saisie de formules, nous avons implémenté deux d'entre eux :

- les *templates*
- l'*overlay*

Pour la méthode des templates, on a mis en place deux boîtes graphiques contenant pour l'une, la liste des opérateurs disponibles dans EMATH, et pour l'autre la liste des symboles grecs. Simplement en cliquant sur l'un des noms, on voit s'afficher l'opérateur correspondant.

Pour pouvoir construire ces boîtes, il faut que l'interface dialogue avec le serveur EMATH pour lui demander la liste des opérateurs qui sont disponibles. Pour efectuer cette opération, nous avons mis au point une fonction qui permet de demander l'évaluation d'une expression au serveur EMATH ; c'est la fonction `evs` : evaluate server.

```
;; evaluate sexpr in context of server
;; send variables of list-of-var to the server before evaluation
;; unbound them after eval
(defmacro evs (list-of-var evs:sexpr &aux
    evs:res
  )
  ;; send vars to server
  (dolist (var list-of-var)
    (send-msg *app-comm-line*
      (print-format String "(setq app::%0 %r1)" var (eval var))))
  ;; evaluate sexpr in server
  (send-msg *app-comm-line*
    (print-format String "%r0" (list 'send-back evs:sexpr)))
  ;; unbound vars in server
  (dolist (var list-of-var)
    (send-msg *app-comm-line*
      (print-format String "(makunbound 'app::%0)" var)))
  ;; retrieve result of evaluation
  (setq evs:res (execute-string (get-next-msg *app-comm-line*)))
  evs:res
)
```

En ce qui concerne la méthode d'*overlay*, celle-ci permet de ne pas avoir recours à la souris et donc d'avoir une saisie beaucoup plus rapide ; on tape de manière continue sur le clavier ce qui évite de perdre du temps en faisant un aller/retour du clavier à la souris pour la saisie de chaque opérateur. D'après le tableau 1.1, on voit qu'il est préférable de tout faire au clavier car une action de pointage à la souris coûte plus cher en temps, ainsi que de passer du clavier à la souris. Bien sûr, la méthode des templates peut être déclenchée au clavier, mais dans ce cas, il faut connaître tous les raccourcis clavier permettant d'appeler les opérateurs, ce qui n'est pas le but recherché.

La figure 5.3 présente l'interface construite, qui n'est qu'un premier prototype.

L'interface a bien sûr la possibilité de définir ses propres opérateurs. Il suffit de faire appel à la fonction `new-operator` et de donner les caractéristiques de ce nouvel opérateur, à savoir : le nom, la priorité, l'arité, la liste des types des fils (s'il n'est pas d'arité n) et une "image" de cet opérateur. En ce qui concerne ce dernier point, plusieurs possibilités s'offrent à nous : cette "image" peut être une entrée dans une police de caractères, un fichier contenant les informations graphiques relatives à cet opérateur...

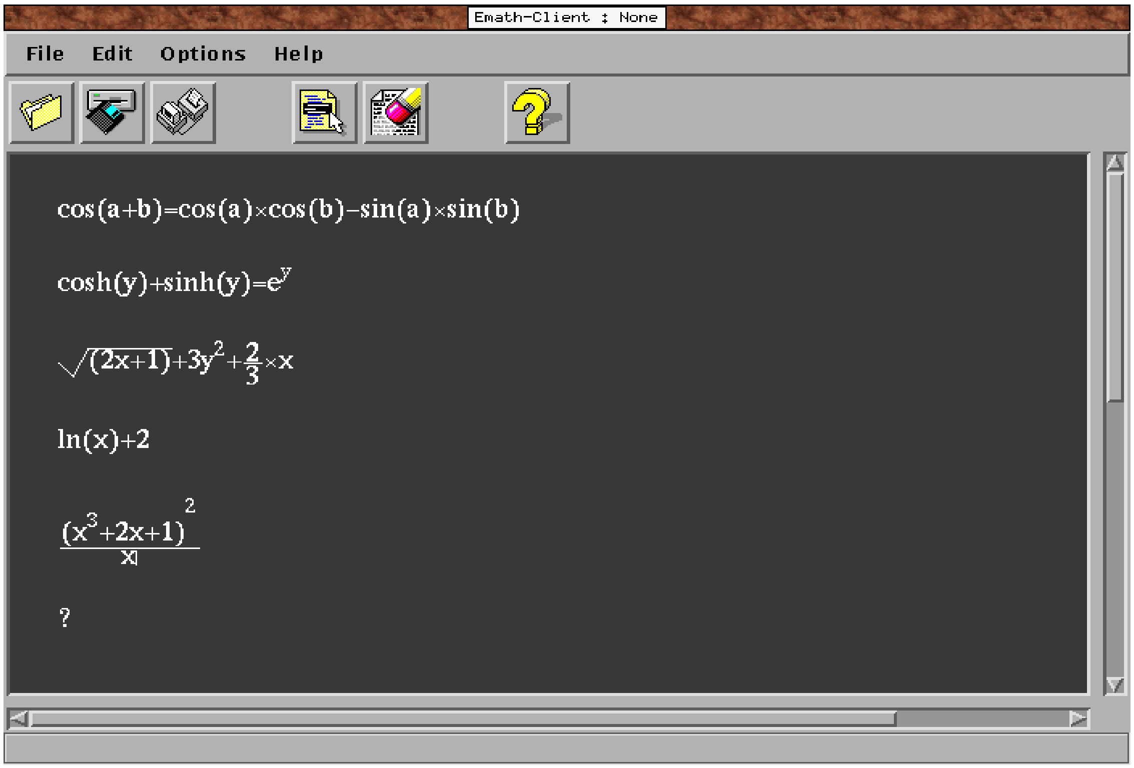

FIG. 5.3 - *L'interface M&M*

Quelques fonctionnalités supplémentaires ont été mises en place dans l'interface afin de faciliter les premiers tests. Parmis ces petits plus, on a :

- sauvegarde et chargement de formules.
- interruption du serveur EMATH pour le débogage.
- début d'aide contextuelle.

5.3 Expérimentations

Nous présentons ici une évaluation qualitative des différentes méthodes qui ont été implémentée. La formule test est : $ax^2 + b$.

5.3.1 Méthode des modèles

Le tableau 5.1 présente la succession des créations et des actions à faire pour construire la formule.

Action	Rendu visuel
Expression initiale	$\boxed{?}$
Modèle *plus*	$\boxed{?}+?$
Modèle *multiplier*	$\boxed{?}?+?$
Lettre a	$a\boxed{?}+?$
Modèle *exposant*	$a\boxed{?}^{?}+?$
Lettre x	$ax^{\boxed{?}}+?$
Chiffre 2	$ax^2+\boxed{?}$
Lettre b	ax^2+b

TAB. 5.1 - *Saisie d'une expression avec la méthode des templates*

D'après le tableau 1.1, on peut calculer le temps que l'on perd lors de l'utilisation de la méthode des templates.

Ce que l'on ne voit pas apparaître dans ce tableau, c'est qu'il faut en plus effectuer des déplacements d'un point d'insertion (?) à l'autre, soit avec la souris, soit par l'appui sur une combinaison de touches. Mais de toute façon, ce sont des opérations supplémentaires qui ne sont pas gratuites.

5.3.2 Méthode de l'overlay

Le tableau 5.2 permet de comparer les actions qui doivent être effectuées par rapport à la méthode précédente.

Pour obtenir le résultat présenté dans le tableau 5.2 avec les actions spécifiées, il faut que l'insertion d'un opérateur se fasse en fonction de sa priorité. En effet, pour avoir

Action	Rendu visuel
Expression initiale	$\boxed{?}$
Lettre a	a
Tape *	$a\boxed{?}$
Lettre x	ax
Tape ˆ	$ax^{\boxed{?}}$
Chiffre 2	ax^2
Tape +	$ax^2 + \boxed{?}$
Lettre b	$ax^2 + b$

TAB. 5.2 - *Saisie d'une expression avec la méthode d'overlay*

une construction correcte de l'arbre, il faut tenir compte bien évidement de la priorité de l'opérateur que l'on insère par rapport à la priorité des opérateurs qui sont déjà présents sur le chemin entre la position courante dans l'arbre et la racine de l'arbre. (Au pire, une hauteur d'arbre à parcourir pour trouver où insérer le nouvel opérateur.)

On voit bien qu'avec la deuxième méthode (celle de l'*overlay*), on a une construction qui est beaucoup plus proche de ce que l'on peut faire naturellement. La suite des actions que l'on fait pour saisir la formule est naturelle, on n'a pas a faire un effort supplémentaire de réflexion pour savoir dans quel ordre effectuer les opérations pour construire la formule désirée.

Conclusion et perspectives

Les interfaces pour le calcul formel ont un rôle important sur la manière d'intéragir avec le système. Malheureusement, les interfaces existantes (*Maple, Mathematica...*) n'utilisent pas des méthodes très ergonomiques en ce qui concerne la saisie et la manipulation des formules. De plus, chaque interface est liée à son propre système de calcul ce qui empêche la réutilisabilité de l'interface pour d'autres systèmes de calcul. Enfin, mis à part ce qui concerne l'affichage des formules (couleur, fontes...), rien n'est paramétrable au niveau des mécanismes de saisie, de manipulation...

Le but de ce stage a été la conception et la réalisation d'un prototype d'interface. L'étude des interfaces existantes et l'expérience acquise dans le projet SAFIR a permis de spécifier un nouveau modèle. Pour réaliser celui-ci, on a utilisé un composant d'affichage de formules (EMATH). Nous avons pu mettre en place une bibliothèque de fonctions implémentant des mécanismes pour la saisie, la sélection et la manipulation des formules.

Le nouveau modèle présente les qualités suivantes :

- Saisie de manière plus naturelle qu'avec les interfaces existantes
- L'architecture adoptée permet d'être indépendant du système de calcul
- Le déplacement spatial est plus intuitif et donc mieux adapté pour les formules que le déplacement syntaxique.

Même si ce modèle semble tenir ses promesses, il reste de nombreuses améliorations à apporter :

- Tester les manipulations à l'aide de la sélection multiple.
- Réaliser la connection effective à plusieurs systèmes de calcul.

Les premiers essais sont encourageants et montrent que le modèle du tableau noir semble être adapté aux besoins actuels et apporte un plus grand confort d'utilisation. Les manipulations sont plus intuitives et ce type d'interface est mieux adapté à la manipulation intensive de formules. Les grandes possibilités de réutilisabilité et de paramétrabilité sont très intéressantes et permettent à la fois une certaine unicité et une spécialisation possibles des interfaces.

La poursuite de ce stage pendant encore trois mois permettra de mettre en place et de tester les fonctionnalités qui n'ont pas pu être encore implémentées.

Annexe A

Exemple de paramétrisation de l'automate

Dans cette annexe nous présentons l'automate et comment le modifier pour avoir un comportement plus adapté à ses propres besoins.

Nous prenons comme exemple un automate le plus simple possible. Cet automate permet de saisir des formules de base constituées de nombres, d'identificateurs (composés de lettres uniquement) et d'opérateurs.

Le schéma 1.1 présente cet automate.

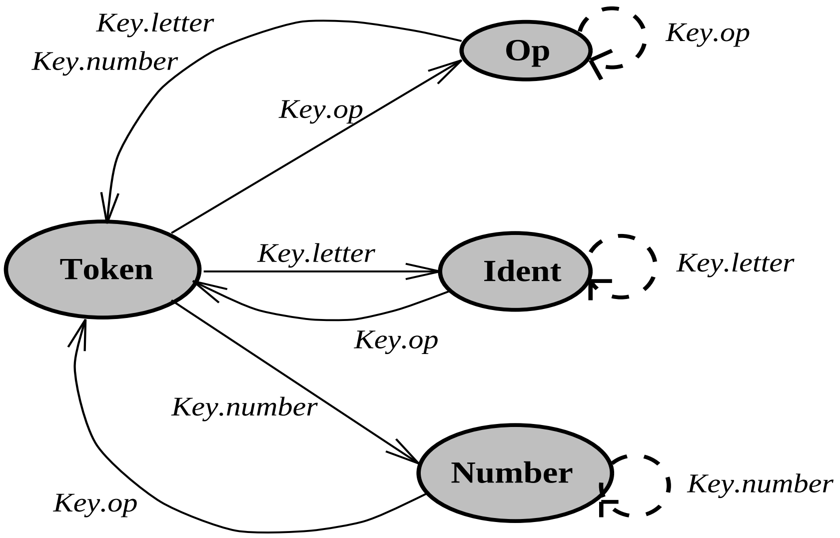

FIG. 1.1 - *Automate de base*

Le code permettant de décrire cet automate est assez simple. Pour chaque état, on a son nom, la touche ou combinaison de touches qui permet de traverser l'état et la fonction à appeler lors de la transition. Nous avons pris une convention au niveau des noms des fonctions pour pouvoir lire l'automate plus facilement. Les fonctions dont le nom est `x-to-y` sont des fonctions qui font passer de l'état `x` à l'état `y` sans *consommer* l'événement qui à permis de franchir la transition (l'evénement est donc retransmis à l'état suivant). Les fonctions du type `x-gob-y` passent de x à y en ne retransmettant pas l'événement.

La manière de décrire les événements qui permettent de franchir une transition a été reprise sur la technique utilisée pour les tables de traduction des événements X. Nous avons apporté une amélioration supplémentaire permettant d'utiliser des expressions régulières pour décrire un ensemble de possibilités (comme pour `[a-zA-Z]`). Cette expression régulière est expansée lors de la pré-compilation de l'automate.

Voici le code permettant de décrire l'automate représenté par la figure 1.1.

```
(defvar emtt-editable "
   (token)         :<Key>[-+*/^]:       (token-to-op)
   (token)         None<Key>[a-zA-Z]:   (token-to-ident)
   (token)         None<Key>[0-9]:      (token-to-number)
   (op)            :<Key>[-+*/^]:       (op-gob-op)
   (op)            None<Key>[a-zA-Z]:   (op-to-token)
   (op)            None<Key>[0-9]:      (op-to-token)
   (ident)         :<Key>[-+*/^]:       (ident-to-op)
   (ident)         None<Key>[a-zA-Z]:   (ident-gob-ident)
   (number)        :<Key>[-+*/^]:       (number-to-op)
   (number)        None<Key>[0-9.]:     (number-gob-number)
  "
;; Translation table for an editable widget
)
```

Nous allons modifier cet automate afin d'avoir un comportement un peu plus *"intelligent"*. Le comportement désiré est le suivant : lorsque l'on saisi un identificateur à la suite d'un nombre, un opérateur multiplication doit venir s'insérer automatiquement entre les deux.

Pour réaliser cela, on crée un nouvel état que l'on va appeler `implicit mult` (multiplication implicite). La fonction associée à cet état se contentera de rajouter un opérateur multiplié entre le nombre qui vient d'être saisi et le nouvel identificateur.

Modification de l'automate pour réaliser ceci :

```
(defvar emtt-editable "
   (token)         :<Key>[-+*/^]:       (token-to-op)
   ...
   (number)        :<Key>[-+*/^]:       (number-to-op)
   (number)        None<Key>[0-9.]:     (number-gob-number)
   (number)        None<Key>[a-zA-Z]:   (number-to-implicit-mult)
   (implicit-mult) None<Key>[a-zA-Z]:   (implicit-mult-to-ident)
  "
;; Translation table for an editable widget
)
```

On obtient ainsi le nouvel automate (figure 1.2).

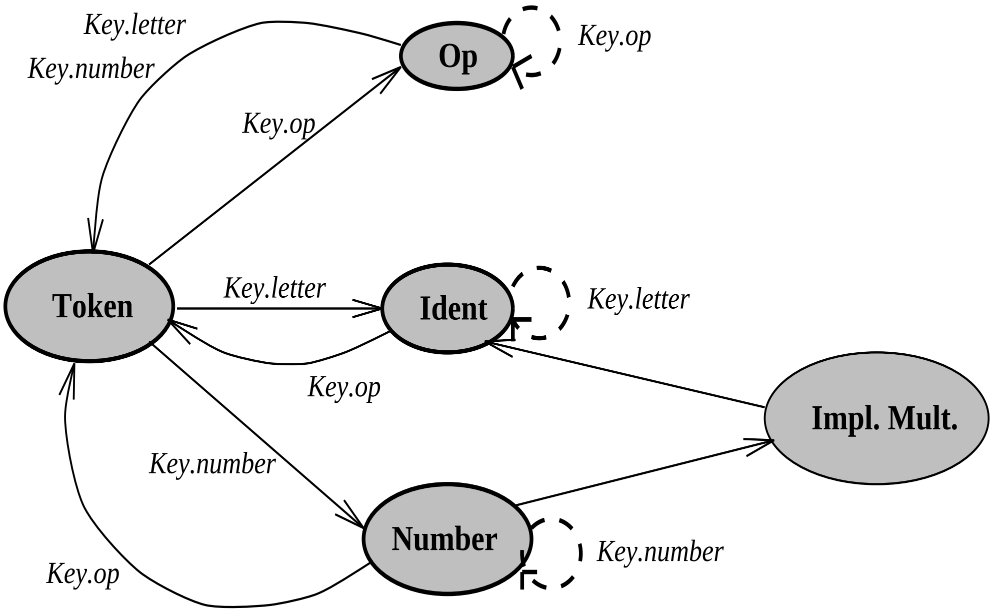

FIG. 1.2 - *Automate de base*

La fonction `number-to-implicit-mult` se contente d'appeler la fonction d'insertion d'un opérateur et de passer dans l'état `implicit-mult`. Comme l'événement a été retransmis, on exécute aussi la fonction `implicit-mult-to-ident` qui fait juste passer dans l'état `ident` en retransmettant l'évenement lui aussi. L'état `ident` va quant à lui ajouter réellement la lettre qui lui a été transmise.

Ci-dessous le code des fonctions `implicit-mult-to-token` et `implicit-mult-to-token` :

```
(defun number-to-implicit-mult (group state event &aux
    (cur-node (get-current-node group))
    (node-value (get-node-value cur-node))
    (op-node (create-node tree:Mult))
    (new-node (create-node tree:Template))
    (tmp-cur-node (%real-current-node cur-node))
  )
  (dont-gobble-event group)

  (insert-node (father-node tmp-cur-node) tmp-cur-node op-node)
  (add-son op-node new-node)
  (set-current-node group new-node)

  (mod-state state 0 'implicit-mult)
)
```

```
(defun implicit-mult-to-token (group state event)
  (dont-gobble-event group)
  (mod-state state 0 'token)
)
```

Dans les paramètres de la fonction `number-to-implicit-mult`, on peut remarquer l'emploi assez peu courant en Lisp de `&aux`. Cette notation est équivalente à un `let` classique. On a préféré l'emploi de `&aux` car l'interprète Klone a été optimisé pour cet opérateur et non pour le `let`.

La fonction `dont-gobble-event` permet de spécifier que l'événement doit être retransmis en sortant de la fonction et la fonction `mod-state` indiquer dans quel état de l'automate on se retrouve en quittant de la fonction.

Annexe B

Construction de l'interface

Klm est un langage de scripts basé sur la boîte à outils Motif (1.2 au minimum) et le format XPM de sauvegarde d'images. Klm utilise Klone, un dialecte de Lisp, dérivé de Wool utilisé pour GWM le window manager paramétrable en Lisp. On peut utiliser Klm dans un mode intéractif ou sous une forme de script exécutable une fois que l'application est finie. A la manière d'un script shell, on peut directement lancer Klm en mettant en tête de fichier : `#!/usr/local/bin/klm`.

Klm fourni une aide en ligne à la manière des commandes `man` ou `apropos`. Par exemple, `(apropos widget)` donne une courte description de toutes les fonctions relatives à la création de widgets ; `(help sujet)` donne toutes les informations sur `sujet`.

```
? (apropos widget)
Label - builds a push button widget
set-subject - associates a subject to a widget
Scrolledlist - a listbox widget contained in a scroller
Separator - A separator widget.
...

? (help Label)
************************************************************************
Label - builds a push button widget

Resources:
- labelString, label, string: the displayed string. default is widget's
    name.
************************************************************************
```

Klm est très pratique pour faire du prototypage et développer ses propres widgets à partir des modules de base existants.

Pour donner une idée du code à fournir pour obtenir l'interface présentée sur la figure

5.3, voici les quelques lignes à écrire :

```
(setq toplevel (Toplevel "client" "Client"))
(setq mainw (MainWindow toplevel "mainw"))
(setq menubar (MenuBar mainw "menubar"))
(setq workarea (Form mainw "workarea"))
```

Les fonctions qui construisent la barre d'aide et la barre de commande

```
(defun create-help-bar ()
  (setq help-bar (Frame workarea "help-bar"))
  (help-bar
    :leftAttachment 'attach_form
    :rightAttachment 'attach_form
    :bottomAttachment 'attach_form)
  (setq help-bar-in (Label help-bar "help-bar-in")))

(defun create-comm-bar ( &aux
    (btn-list '(
        ("open.xpm"   (show dialog:load-file-w)   "open-inv.xpm"    t)
        ("save.xpm"   (save-cb)                   "save-inv.xpm"    t)
        ...
        ("gomme.xpm"  (undo)                      "gomme-inv.xpm"   t)
        (""           ()                          ""               ())
        ("help.xpm"   (query-for-help toplevel)   "help-inv.xpm"    t)))
    btn-i)
  (setq comm-bar (RowColumn workarea "comm-bar"))
  (comm-bar
    :leftAttachment 'attach_form
    :rightAttachment 'attach_form
    :topAttachment 'attach_form)
  (dolist (i btn-list)
    (if
      ;; if we've got a pixmap and an associated callback
      (not (= (get i 0) ""))
      (progn
        (setq btn-i (PushButton comm-bar (get i 0)))
        (btn-i
          :marginWidth 1
          :labelType 'PIXMAP
          :labelPixmap (get i 0)
          :alignment 'ALIGNMENT_CENTER)
        (if (not (get i 3 ()))
          (btn-i
            :sensitive 'False))
```

```
        (callback btn-i 'activate (get i 1)))
      ;; else, we must place a separator
      (setq btn-i (Label comm-bar "Separator"))))))

(create-help-bar)
(create-comm-bar)
```

On définit les éléments de la zone même de travail :

```
(setq scrollarea (ScrolledWindow workarea "scrollarea"))
(scrollarea
  :leftAttachment 'attach_form
  :rightAttachment 'attach_form
  :bottomAttachment 'attach_widget :bottomWidget help-bar
  :topAttachment 'attach_widget :topWidget comm-bar)

(setq draw (DrawingArea scrollarea "draw"))
```

Définition de la barre de menu et des menus :

```
(setq file (PulldownMenu menubar "file"))
(setq edit (PulldownMenu menubar "edit"))
(setq options (PulldownMenu menubar "options"))
(setq help (PulldownMenu menubar "help"))
(menubar :menuHelpWidget help)

(PushButton file "new")
(callback file.new 'activate '(new-file))
(Separator file "-")
(PushButton file "open")
(callback file.open 'activate '(load-cb))
...
(PushButton help "version")
(callback help.version 'activate '(show help-box:help-on-version))
```

Pour construire toute l'application et afficher les différents widgets :

```
(build toplevel)
```

Bibliographie

[Avi90] Avitzur (Ron). – Suggestions for a friendlier user interface. *In : LNCS 429, DISCO'90*, éd. par Imola (A.). pp. 282–283. – Springer-Verlag.

[Bou93] Bouthors (Vincent). – *Un interprète embarqué dans un système de gestion d'interfaces homme-machine.* – Thèse, Université de Nice Sophia Antipolis, décembre 1993.

[BW87] Bonadio (Allan) et Warren (Erick). – *Theorist Reference Manual.* – Prescience Corporation, 814 Castro St, San Francisco, CA, 94114, 1987.

[CFZ93] Chabrier (Bruno) et Franchi-Zannettacci (Paul). – *Interface graphique par délégation sémantique.* – Rapport technique n93-18, I3S, mars 1993.

[DGS94] Dalmas (Stéphane), Gaëtano (Marc) et Sausse (Alain). – Distributed computer algebra: the central control approch. *In : PASCO'94*, pp. 104–113. – Linz, Austria, septembre 1994.

[DGS95] Dalmas (Stéphane), Gaëtano (Marc) et Sausse (Alain). – The central control: an introducttion. *In : Proceedings of the Posso workshop on software*, éd. par Jean-Charles Faugère, Joel Marchand (Renaud Riobbo), pp. 183–199. – Paris, France, mars 1995.

[DW90] Doleh (Y.) et Wang (P.S.). – A system independant user interface for an integrated scientific computing environment. *In : ISSAC'90*, pp. 88–94.

[Fra93a] Frame Technology Corporation. – *FrameMaker : Manuel d'utilisation X/Motif Version*, septembre 1993, 4ème édition.

[Fra93b] Frame Technology Corporation. – *FrameMaker : Prise en main X/Motif Version*, septembre 1993, 4ème édition.

[Kaj92] Kajler (Norbert). – Cas/pi: a portable and extensible interface for computer algebra systems. *In : Proc. of ISSAC'92.* pp. 376–386. – Berkeley, USA, juillet 1992.

[Kaj93] Kajler (Norbert). – *Environnement graphique distribué pour le calcul formel.* – Thèse, Université de Nice Sophia-Antipolis, mars 1993.

[KS94] Kajler (N.) et Soiffer (Neil). – A survey of user interface for computer algebra systems. *Academic Press*, juin 1994.

[Leo86] Leong (B.L.). – Iris: Design of a user interface program for symbolic algebra. *In : Proceedings of the 1986 Symposium on Symbolic and Algebraic Computation*, pp. 1–6.

[Mic92] Microsoft Corporation. – *Editeur d'équations: Guide de l'utilisateur*, 1991-92.

[Min63] Minsky (Marvin). – *A Proposal for a Mathematical Manipulation - Display System.* – Rapport technique, MIT, Artificial Intelligence Project, Project MAC, novembre 1963.

[Nah88] Nahaboo (Colas). – *D'un paragrapheur à un éditeur syntaxique et graphique pour le langage ESTEREL.* – Thèse, Universite de Nice Sophia-Antipolis, novembre 1988.

[Soi91] Soiffer (Neil). – *The Design of a User Interface for Computer Algebra Systems.* – Phd thesis, Berkeley, University of California, avril 1991.

[SS86] Smith (Carolyn) et Soiffer (N.). – Mathscribe: A user interface for computer algebra systems. *In : Proceedings of the 1986 Symposium and Algebraic Computation*, pp. 7–13.

[VL87] Valentin (A.) et Lucongsang (R.). – *L'Ergonomie des logiciels.* – Edition de l'anact, 1987, *Collection outils et méthodes*, 118p.

[YW87] Young (Douglas A.) et Wang (Paul S.). – Gi/s: A graphical user interface for symbolic computation systems. *In : Journal of Symbolic Computation.*

Une interface pour le calcul formel

Résumé :

Dans le domaine du calcul formel, la plupart des efforts ont été concentrés sur les méthodes algorithmiques, les langages de programmation ... mais très peu en ce qui concerne les interfaces. Pourtant, nous avons pu nous rendre compte que celles-ci sont importantes car une déficience au niveau de l'interface entraîne un manque de productivité et les rend difficiles à utiliser.

Après une étude des interfaces pour le calcul formel, nous avons constaté que même si l'on dispose actuellement d'outils très puissants pour le calcul formel, il n'y a pas d'interface permettant une saisie intuitive et une manipulation aisée des formules. De plus, les interfaces sont faites pour le système de calcul auquel elles sont rattachées et donc on ne peux pas s'en servir pour adresser d'autres moteurs de calcul.

Le but du stage a donc été de spécifier et d'implémenter un prototype d'interface permettant de se connecter à un ou plusieurs systèmes de calcul formel (comme Maple, Mathematica, Reduce...) et d'expérimenter des mécanismes pour l'édition et la manipulation de formules. Ce prototype utilise un nouveau modèle d'interface type tableau noir, qui est entièrement paramétrable.

Mots clés : Calcul Formel, interface homme-machine, interface graphique.