Cahiers GUTenberg

APPELER TEX COMME UNE FONCTION

Jonathan FINE

Cahiers GUTenberg, nº 42 (2003), p. 26-37.

<http://cahiers.gutenberg.eu.org/fitem?id=CG_2003___42_26_0>

Appeler TEX comme une fonction*

Jonathan FINE

Coldhams Lane
Cambridge, CB1 3HY
Royaume-Uni
Courrier électronique : jfine@activetex.org
web : httpx://www.activetex.org

Résumé. TEX est traditionnellement exécuté comme un simple programme. Pourtant, TEX peut aussi être exécuté comme un démon et être appelé comme une fonction par une interface. Cet article décrit les perspectives et problèmes découlant de cette nouvelle méthode d'utilisation de TEX.

Mots-clés : Démon, TEX interactif.

Abstract. *Traditionally, TEX is run as a batch program. However, TEX can also be run as a daemon, with a callable function interface. This talk describes the opportunities and problems that follow from this new way using TEX.*

Keywords: *Daemon, interactive TEX.*

1 Le démon TEX

TEX est un programme de traitement de texte alliant fiabilité et haute qualité qui fonctionne sur une large variété de matériels. Il est remarquablement rapide pour la composition, une fois lancé. Cependant, une bonne fraction de seconde lui est nécessaire pour initialiser ses structures de données, en chargeant le fichier d'un format pré-compilé. Le démon[1] TEX est un moyen d'éviter ce temps d'initialisa-

*Cet article est la traduction en français de *TEX as a Callable Function*. Ce texte est paru dans les actes de la conférence EuroTEX 2002 (p. 26-30), qui s'est tenue à Bachotek (Pologne) en avril et mai 2002, et a été également présenté à la conférence TUG 2002, à Trivandrum (Inde), en septembre 2002. La traduction a été réalisée par Jean-Michel Hufflen (hufflen@lifc.univ-fcomte.fr) ; elle est publiée avec l'aimable accord de l'auteur et de Tomasz Przechlewski, rédacteur en chef des actes d'EuroTEX 2002.

1. N.d.T. : pour la traduction des termes purement techniques, nous avons le plus souvent suivi la terminologie donnée dans le *Dictionnaire d'informatique anglais/français*, de Michel Ginguay, paru en 2001 aux éditions Dunod, sauf dans quelques cas où nous avons conservé le terme anglais car aucune traduction en français ne s'est réellement imposée.

tion, c'est-à-dire de le supprimer de la boucle traditionnelle d'édition, compilation et prévisualisation. Réaliser cette suppression rend possibles *Instant Preview* et d'autres applications interactives de TEX.

Un *démon* ou *service* est un programme qui est exécuté continuellement dans l'arrière-plan, attendant d'être sollicité si besoin est. Beaucoup de services liés à Internet, tels que FTP ou HTTP, sont fournis par des démons. Sur requête, un démon fournit des données, ou un service tel qu'une impression. Par nature, beaucoup de démons n'ont pas d'état. En d'autres termes, une requête FTP n'affecte pas les résultats d'une autre. Les *cookies*[2] fournissent des états à HTTP, qui n'en aurait pas sinon. Pour quelques démons tels que le serveur d'une base de données, l'une des tâches essentielles est d'enregistrer les changements d'états de façon sûre.

En exécutant TEX comme un démon, nous évitons le coût du démarrage. Pour la composition d'un seul alinéa, cela peut donner un facteur d'accélération de 30 à 50. C'est précisément ce gain qui permet d'utiliser TEX comme un moteur de composition d'applications interactives. Il est toutefois nécessaire de prendre des précautions. Si certaines commandes sont données comme aliments à TEX, elles peuvent lui causer une indigestion sévère ou des désagréments pires.

Par exemple, en *plain TEX*, la commande `\end` va dans la plupart des cas provoquer l'arrêt de TEX (pour LATEX, la commande est `\stop`). C'est souhaitable, tôt ou tard, dans un simple programme[3], mais pas dans un démon. Cet exemple est trivial, mais il montre que des entrées inappropriées peuvent endommager ou causer l'arrêt du démon TEX.

Quelques exemples sont plus subtils. En tout premier, la capacité de TEX est finie et peut être dépassée. Donnons-lui trop de noms de séquences de contrôle, au moyen de `\csname`[4], et l'espace d'adressage dispersé de TEX va arriver à épuisement. Ce n'est en général pas un problème lorsqu'on traite un seul document. Mais si nous voulons que le démon TEX soit actif tout un jour, traitant des fragments extraits de centaines de documents, l'impossibilité pour TEX de récupérer la place des noms des séquences de contrôle inutilisées pourrait constituer un problème (voir les bogues nos 422 et 493 dans le rapport des erreurs de TEX et la section *Surprises* de [3]).

2. N.d.T. : ce terme a été conservé tel quel car la traduction littérale (petits gâteaux) a relativement peu de sens. Il s'agit d'un mécanisme permettant à un serveur web de conserver des informations spécifiques au client sur la machine client.

3. N.d.T. : nous appelons « simple programme » un programme dont l'exécution est clairement confinée : il est lancé, rend le service que l'on attend de lui, et se termine.

4. N.d.T. : nous signalons à l'intention du lecteur francophone que toutes les commandes de TEX sont décrites en français dans *La maîtrise de TEX et LATEX*, de Thomas Lachand-Robert, paru en 1995 aux éditions Masson. On y trouvera en particulier la description de toutes les commandes auxquelles Jonathan Fine fait allusion.

L'exemple le plus subtil que je connaisse concerne la coupure des mots. La commande `{\hyphenation{su-per-cal-i ... }}` ajoute un mot au dictionnaire des exceptions (pour le langage courant). Ce dictionnaire est toutefois global. Les exceptions ne disparaissent pas à la fin du traitement d'un groupe. Du coup, comme ces exceptions aux coupures sont ajoutées durant tout un jour, l'état de TEX change lentement. Un alinéa qui était découpé en lignes d'une certaine façon le matin peut être découpé différemment l'après-midi. TEX, à juste titre, a la réputation de donner des résultats identiques sur des entrées identiques. Pour que le démon TEX préserve cette qualité admirable, il faut interdire l'ajout de nouvelles coupures exceptionnelles, ou trouver un moyen de revenir à l'état précédant un tel ajout, ou encore trouver une autre solution.

Ce sont toutefois des problèmes mineurs. TEX a beaucoup de qualités admirables comme démon. Il ne se « plante » pas, ne fabrique pas de fichier core, ne se laisse pas aller à une fin brutale. Il se comporte de façon extrêmement prédictible (même s'il nous surprend quelquefois). Il ne dégrade pas la mémoire. Avec des entrées correctes, il composera des pages toute la journée et sera aussi dispos au crépuscule qu'il l'était à l'aube.

Bien que conçu comme un simple programme, il peut être aussi exécuté comme un démon : c'est une preuve remarquable de la sûreté de sa conception et de sa programmation.

2 Appeler TEX comme une fonction

TEX prend comme entrée un flux de texte et sa sortie est un flux de pages, codées dans le format dvi (d'autres données sont bien sûr présentes, telles que les fontes, les données relatives aux coupures et les macros). Le fichier .dvi de sortie consiste en un préambule, suivi de quelques pages et d'un postambule. Chaque fonte qui apparaît dans le fichier .dvi est définie deux fois, d'abord immédiatement avant sa première utilisation et une seconde fois dans le postambule [5]. Presque tous les programmes lisant du dvi vont immédiatement au postambule pour récupérer la liste des fontes (le prévisualisateur de Tom Rokicki pour la machine NeXT est une exception : il peut commencer la prévisualisation à partir de la première page). Ils récupèrent à cette même place un pointeur sur la dernière page, puis à partir de là, un pointeur sur la page précédente et ainsi de suite. Cette opération achevée, ils sont en position de réaliser des accès aléatoires aux pages du fichier .dvi. Supposons qu'un tel programme doive trouver et traiter la page 20 : si le postambule n'était pas disponible, il devrait traiter toutes les pages de 1 à 19, d'une part pour trouver

5. N.d.T. : les lecteurs francophones peuvent trouver davantage de détails à propos du format dvi dans l'article de Denis B. Roegel, « Les formats de fichiers DVI, GF, TFM et VF : que contiennent-ils et comment les visualiser ? », *Cahiers GUTenberg*, nº 26, mai 1997, p. 71-95.

la page 20, d'autre part pour être sûr de récupérer les définitions précédentes de toutes les fontes.

C'est clair que si TEX est exécuté comme un démon, il n'est pas possible d'attendre le postambule avant de répondre à la requête d'un utilisateur. Au lieu de cela, nous faisons écrire TEX dans un tube nommé ou quelque chose de similaire et nous avons placé à l'autre bout du tube un programme utilitaire qui analyse le train des données dvi de sortie puis envoie les pages à la destination adéquate.

C'est exactement ce qu'effectue le programme dvichop de l'auteur. Il coupe le train de données de sortie en petits fichiers .dvi et envoie sur option un signal au processus d'origine. Cette décision de conception permet aux logiciels de dvi existants de fonctionner avec le démon TEX au coût relativement supportable de traiter deux fois chaque octet du dvi.

Un autre avantage de cette approche : elle permet de spécifier le comportement de l'appel de TEX comme une fonction. Dans la section précédente, nous avons appris que le démon TEX devrait avoir un état et qu'il était indispensable que chaque utilisation du démon TEX le laisse dans le même état. Cela impose des conditions sur l'alimentation que nous donnons au démon. L'état fixé du démon TEX peut être installé par un fichier de format pré-compilé.

Nous pouvons maintenant définir ce que sont des entrées valides pour le démon TEX : elles peuvent produire éventuellement des sorties dvi, mais ne changent pas l'état du démon TEX. Les commandes de groupement déjà fournies par TEX auront un long chemin à parcourir pour valider de telles entrées. Nous pouvons aussi définir la sortie du démon TEX pour toute entrée valide. Par simplicité, nous supposons que TEX exécute la commande `\input` sur le fichier, avec certaines macros fixées qui s'exécutent avant et après la commande `\input`.

La sortie de la fonction TEX est donc le fichier .dvi produit, depuis l'état initial défini par le fichier de format jusqu'à la fin du traitement de la donnée valide et l'émission d'une commande `\end`, `\bye` ou `\stop` (une autre fin étant un arrêt brutal de TEX).

Nous venons de définir, sans référence à l'implantation, le comportement de l'appel de TEX comme une fonction. La définition ci-dessus donne une implantation. Utiliser le démon TEX (par exemple avec dvichop) en donne une autre qui peut être plus rapide sur des fichiers divisés en pages.

Ce qui devrait être clair, c'est que le fichier de format pré-chargé et la création d'entrées valides sont cruciaux pour toute mise en œuvre du démon TEX. À vrai dire, quatre approches existent pour le problème des entrées valides. La première est la définition originale, c'est-à-dire que l'état de TEX n'est pas modifié. Cependant, cette définition ne peut détecter des entrées incorrectes qu'après coup, c'est-à-dire trop tard. En outre, elle ne donne pas une interface pour laquelle un utilisateur du démon TEX sache écrire du code.

La seconde approche est de s'assurer que toute entrée, quoi qu'il arrive, peut être traitée sans changer l'état de TEX. Ordinairement, l'entrée de TEX a un accès direct à toutes les commandes primitives et toutes les macros de TEX. Cette approche implique l'insertion d'une couche de macros entre le fichier d'entrée et l'exécution des macros. C'est similaire à la protection offerte par le noyau d'un système d'exploitation actuel, qui ne permet pas aux programmes d'un utilisateur d'accéder directement au matériel.

C'est la méthode d'*Active TEX* [1], qui rend actifs tous les caractères. Les seules commandes que l'entrée utilisateur peut directement accéder sont celles qui sont liées aux caractères. Ces commandes, à leur tour, vont produire et exécuter des séquences de contrôle. Néanmoins, ce n'est pas l'utilisateur qui régit cette production de séquences de contrôle, mais la couche système des caractères actifs.

La troisième approche est d'utiliser un programme spécialement pensé pour filtrer les entrées et les traduire en des appels standard de TEX, les mauvaises étant éliminées. Utiliser un *script* XSLT pour générer du code LATEX est un exemple de cette approche. L'entrée ne crée pas directement des appels à TEX et l'instance de LATEX ainsi créée peut être étroitement contrôlée. Dans la plupart de ces cas, on devrait donner — en principe sinon dans les faits — une spécification formelle pour les sorties possibles. Elle serait un contrat entre le créateur d'entrées pour TEX et le fournisseur de macros de TEX.

Une remarque : selon l'auteur, bien que LATEX possède beaucoup de capacités et une large gamme de bibliothèques, ce n'est pas un langage adapté à ce propos. C'est parce qu'il a beaucoup d'exceptions et parce qu'il n'a pas été conçu pour des entrées et sorties générées par un programme prenant en charge l'aspect interactif. Par exemple, beaucoup de ses facilités sont reliées à des changements dans le code des catégories des caractères. En outre, le caractère « [» a parfois un rôle spécial. L'auteur, qui pense lui-même être un expert, tombe de temps en temps dans de tels pièges.

Utiliser XSLT pour produire du code LATEX est un raccourci appréciable pour produire de l'écrit à partir de XML. Néanmoins, même les supporters les plus dévôts en conviendront sûrement, faire de LATEX un format d'interface pour un navigateur web ou un traitement de texte WYSIWYG [6] serait un triomphe de l'inertie sur le développement sain.

La quatrième approche est d'ignorer le problème et de fournir un moyen pour faire repartir le démon TEX en cas d'ennui. *Instant Preview* utilise largement cette approche, en exécutant TEX en `\errorstopmode` et en utilisant une macro `\outer` judicieuse pour stopper la propagation d'erreurs en dehors du niveau de l'utilisa-

6. N.d.T. : *What You See Is What You Get*, « ce que vous voyez est ce que vous obtenez ». Cette expression caractérise les systèmes interactifs de mise en page, tels que Microsoft Word.

teur. Bien sûr, si l'entrée de l'utilisateur contient `\global\let\let\undefined` (ce qui n'est pas une entrée sensée), alors ce dernier va récolter ce qu'il a semé.

Finalement, bon nombre de problèmes techniques sont reliés à l'implantation de la fonction TEX. Dans cet article, nous voulons simplement en noter trois. L'appel de la fonction va envoyer une chaîne à TEX et s'attendre à recevoir du dvi en retour. Le premier problème est de ne pas effectuer l'instruction de retour de la fonction tant que le dvi n'est pas accessible ou, en d'autres termes, attendre jusqu'à ce que TEX ait fini. Le deuxième est de ne pas provoquer de blocage ou, en d'autres termes, éviter que TEX et l'appel de la fonction attendent chacun une entrée de la part de l'autre. Le troisième est de gérer les conflits ou, en d'autres termes, les requêtes se chevauchent dans le temps. Ce sont là des problèmes standard en programmation client-serveur et les solutions sont bien connues même si elles ne le sont pas dans la communauté de TEX.

3 Composition en accès aléatoire

Quelques développeurs, outre moi-même, ont écrit du logiciel qui donne à la personne éditant un document un retour visuel plus ou moins immédiat. Tous, nous avons affronté le même double problème. Comment découper dans le document une région à composer et comment initialiser TEX pour qu'il puisse correctement traiter cet extrait. Il y a des problèmes importants dans la composition en accès aléatoire, une notion que nous allons maintenant définir.

Beaucoup d'utilisateurs de TEX savent ce qu'est une prévisualisation de TEX en accès aléatoire. Cela signifie charger un fichier .dvi rapidement et accéder rapidement à n'importe quelle page dans le document prévisualisé. Rappelons-nous la section précédente : un fichier .dvi a une structure particulière, spécifiquement conçue pour la mise en œuvre de telles opérations. Composer en accès aléatoire, c'est pouvoir aller en tout point dans le document source et composer rapidement une région entourant ce point. Les sauts de page (ainsi que les numéros de page) présentent des problèmes particuliers. Néanmoins, le temps consacré à leur résolution n'est pas si important en général pour l'utilisateur et nous ignorerons donc ce problème.

Dans une forme étendue de la composition en accès aléatoire, c'est non seulement le point dans le document qui est aléatoire, mais aussi le choix du document lui-même. C'est ce genre de tâche que doit accomplir un navigateur web. Plus loin dans cette section, nous verrons comment cela induit une nouvelle gamme de problèmes.

Un document LATEX valide a une structure comprise par les macros de LATEX. Pourvu que le document ne soit pas trop insolite, il est possible qu'un autre programme

— par exemple une collection de macros d'emacs — divise le document en blocs qui peuvent être composés individuellement. Néanmoins, un tel logiciel est fragile. Des macros définies par l'utilisateur, mal tapées au clavier ou même bien conçues, peuvent briser un tel système. Les scripts Perl qui convertisent LaTeX en HTML font face à des problèmes similaires, avec lesquels leurs utilisateurs sont familiers.

Ceci nous amène au deuxième problème, qui est d'établir proprement le contexte dans lequel le fragment de document peut être composé. Connaître les bons numéros de sections et de théorèmes est une partie de ce problème. Savoir dans quel contexte (résumé, note en bas de page, alinéa) composer en est l'autre partie.

La solution préférée de l'auteur à ce problème est de confier la responsabilité de sa résolution au document en cours d'édition. En d'autres termes, on écrit un script, fondé sur une version de LaTeX vers HTML qui ajoute à ce document ce que nous pouvons appeler un point d'assurance (en escalade, un *point d'assurance*[7] est utilisé par un grimpeur pour attacher une corde). Chaque point d'assurance devrait comprendre les numéros de sections, de théorèmes, etc. aussi bien qu'une information à propos du contexte de composition.

Il n'y a pas besoin de ranger les données du point d'assurance dans le document lui-même. Elles peuvent être rangées dans le fichier `.aux` (et être rendues disponibles à LaTeX par une scrutation au moyen de la commande `\csname`). Dans cette optique, toutes les informations qui doivent être ajoutées au document sont des commandes telles que `\belay{27}`. Ce qui peut être fait par un script Perl. En plus, `\usepackage{belay}` dans le préambule va provoquer l'écriture de l'information du point d'assurance dans le fichier `.aux` lors de l'exécution d'une composition initiale. À présent, belay.sty est en fait éphémère. Des remerciements sont dus à Simon Dales, qui m'a suggéré le terme « point d'assurance » et à Johann Andersson, pour avoir partagé avec moi un prototype qu'il a réalisé en suivant des lignes semblables.

À présent, LaTeX est installé pour composer en simple programme et pour composer en accès aléatoire. Ceci cause des problèmes variés. Par exemple, dans le fichier de style pour un article, la commande `\maketitle` se redéfinit en la commande `\relax`. Il peut exister d'autres surprises.

Venons-en à une autre catégorie de problèmes. Chaque document LaTeX a un préambule, et même quand deux articles sont destinés au même journal, ils ont très souvent des préambules différents. Même si deux préambules diffèrent seulement par l'utilisation d'un paquetage, ceci présente un problème car actuellement, LaTeX ne permet le chargement de paquetages que dans le préambule, et non après le lancement de la composition. Quand deux documents numérotent les théorèmes

7. N.d.T. : « *belay* » en anglais, d'où les noms donnés par Jonathan Fine aux commandes et paquetage correspondants.

de façons différentes, ce sont des macros définies par les utilisateurs qui sont différentes, ce qui crée des problèmes supplémentaires.

À la lumière de cela, l'auteur croit qu'il n'est pas viable pour deux documents LaTeX en accès aléatoire de partager le même démon TEX. L'auteur croit également qu'avec quelques changements judicieux, LaTeX peut être utilisé avec succès pour la composition en accès aléatoire d'un document unique. Quelques-uns de ces changements sont reliés au traitement d'erreurs. Par exemple, `\scrollmode` puis `\section` sans aucun argument produit un caractère «] » isolé dans le fichier .dvi de sortie.

4 Macros utilisables avec le démon TEX

TEX est inutilisable sans macros car ses commandes primitives sont… disons… primaires. Un format de macros tel que *plain* ou LaTeX effectue quelques opérations. Voici quelques-unes d'entre elles : (1) il charge les fontes et les motifs de coupure ; (2) il définit une syntaxe d'entrée en vigueur ; (3) il ajuste des paramètres typographiques tels que les dimensions et fournit des commandes pour changer ces valeurs ; (4) il rend disponibles des commandes pour la composition des mathématiques ; (5) même chose pour les tables ; (6) il définit une routine de sortie ; (7) il prend soin de la numérotation des pages, sections, équations, etc. ; (8) il transcrit les informations pour l'index et la table des matières.

Plain et LaTeX ont été écrits pour une utilisation de TEX en simple programme. L'utilisateur crée un document qui est soumis au compilateur TEX. Ensuite TEX retourne, espérons-le, un fichier .dvi et un fichier .log de trace. L'utilisateur les étudie ensuite tous les deux. Le fichier de trace consigne les erreurs de syntaxe (par exemple, des séquences de contrôle mal orthographiées) et les difficultés de la composition (par exemple, des boîtes qui débordent). Hormis le support et le délai d'exécution, la situation est la même que lorsque des cartes perforées étaient soumises à un processeur central.

Le démon TEX procède d'une installation complètement différente. Le programme TEX est déjà en exécution et nous serions fort aise si l'entrée de l'utilisateur laissait le démon dans l'état où nous aimerions le trouver. La solution robuste est d'écarter les erreurs de l'utilisateur, en particulier celles qui endommagent le démon TEX. Comme cela a été remarqué précédemment, des macros de TEX peuvent le faire, mais d'autres méthodes sont peut-être meilleures.

À une session de questions et réponses, Piet van Oostrum interrogea Don Knuth au sujet du langage de programmation des macros de TEX [5, p. 648-649]. Le lecteur est encouragé à lire la totalité de sa réponse et, à vrai dire, toutes les sessions de questions et réponses. Nous résumons ici les points présentant un intérêt par-

ticulier : (1) pour TEX, Don a voulu éviter d'introduire « encore un autre langage de programmation, presque le même » ; (2) beaucoup de caractéristiques ont été ajoutées uniquement « après des réclamations à cor et à cri » de la part d'utilisateurs ; (3) les utilisateurs voulaient porter des opérations de bas niveau à un plus haut niveau ; (4) Don s'attendait à ce que des applications particulières soient réalisées par des changements dans le code compilé ; (5) Don voulait simplement écrire un langage de composition et non un langage de programmation ; (6) il a aussi dit : « S'il y avait eu un langage interprétatif simple et en usage dans d'autres systèmes, j'aurais naturellement jeté mon dévolu sur lui pour TEX. »

J'ai passé plusieurs centaines d'heures à écrire des macros ingénieuses de TEX pour réaliser des opérations de haut niveau (comme analyser du texte SGML) et la collectivité des auteurs de gros paquetages de macros en a passé probablement d'aussi longues à ce genre de tâche. J'ai appris à m'accomoder des limitations du langage, à utiliser au mieux les caractéristiques qu'il fournit. Quelques-unes de ses ruses de programmation sont astucieuses et intelligentes. Néanmoins, je pense qu'il est temps de changer.

Une des principales raisons est que du code de macros ingénieuses de TEX ne peut être utilisé qu'avec TEX (ou ses successeurs) et, réciproquement, du code ingénieux — ou simplement solide et fiable — dans un autre langage ne peut pas être utilisé à l'intérieur de TEX (quoiqu'il existe quelques contournements). Une autre raison est que d'autres langages peuvent être plus efficaces, tant pour le programmeur que pour la machine.

Voici un exemple. Analyser syntaxiquement une chaîne de texte c'est décortiquer sa structure, la séparer en unités arrangées d'une certaine façon. Un analyseur syntaxique d'une langue naturelle trouvera les sujet, objet et verbe dans une phrase. L'analyse syntaxique est une activité non triviale, de laquelle des traitements futurs dépendent. LATEX contient un analyseur. Mais comme il est écrit avec les macros de TEX, il ne peut pas être partagé avec d'autres applications.

Il existe une alternative aux macros de TEX que Don n'a pas mentionné dans sa réponse. C'est peut-être qu'il la considère comme admise. La programmation documentée est un exemple. Un programme client (WEAVE) y prend un document que TEX ne peut pas comprendre et en dérive un fichier d'entrée pour TEX. Quelques scripts XLST sont un autre exemple.

TEX a été écrit dans les années mille neuf cent soixante-dix et quatre-vingt et, depuis, sont apparus de nouveaux langages interprétatifs, quelques-uns d'entre eux étant largement utilisés : Perl, Python et Ruby, Java et JavaScript, Scheme et Guile, Visual Basic et C#.

Est-ce que l'un d'entre eux, selon les termes de Don, est « un langage interprétatif simple et universel […] en usage dans d'autres systèmes » ? Mais est-ce un jugement de Pâris ? Comment pouvons-nous en utiliser un sans offenser tous les

autres ? Sans amorcer une guerre de langages ? C'est un problème difficile et comme tous les problèmes humains, sa solution requiert de la bonne volonté, des bonnes idées et une dose de sagesse.

L'auteur suggère que les macros de TEX ne soient utilisées que là où elles seules sont opérationnelles, ou bien là où l'efficacité le requiert. Utiliser C ou C++ pour des modules externes tels qu'un analyseur de XML, des langages de scripts pour le contrôle du style et du placement, ainsi que pour du code spécifique à une application. Une application actuelle type pourrait aujourd'hui utiliser Perl pour extraire des enregistrements d'une base de données et les envoyer à TEX pour composition. Sous ce nouveau schéma, l'application serait la même, sauf que le train d'entrées de TEX serait écrit en utilisant un module d'interface en Perl plutôt que directement.

Nous avons au moins un bon exemple à suivre. La boîte à outils graphiques Tk a été développée par John Oosterhout comme un compagnon au langage de script Tcl. Dès lors que les modules d'interface pour Perl et Python ont été écrits, cela permet à ces langages d'appeler des fonctionnalités de Tk. Bien que Perl, Python et Tcl aient des syntaxes différentes, tous ces langages peuvent utiliser les mêmes méthodes d'interface avec Tk. Peut-être quelque chose de similaire pourrait être fait avec TEX.

5 Logiciels dvi utilisables avec le démon TEX

Des applications interactives introduisent de nouvelles requêtes sur les fichiers .dvi, et sur les programmes utilisés pour les traiter. L'étendue des interactions avec la page prévisualisée est vaste mais, à présent, tout ce qui est pris en charge, ce sont les hyperliens ainsi que le marquage du texte dans quelques cas.

En effet, la plupart des programmes pour afficher du dvi à l'écran sont des prévisualiseurs tant par le nom que la fonction. Une pré-vue est une description de quelque chose qui n'est pas encore définitif. Dans notre cas, elle décrit une page composée que nous pouvons choisir ou non d'imprimer. Néanmoins, dans beaucoup d'applications interactives, ce qui est affiché n'est pas une pré-vue, mais l'objet lui-même. En effet, il est courant, dans les journaux, d'afficher la copie d'écran d'un site web, afin de fournir une prévisualisation imprimée de l'objet visé, c'est-à-dire du site web.

Comme processeur dvi, dvichop n'est ni compliqué ni particulièrement spécialisé. Avant de l'écrire, j'ai regardé le source d'un grand nombre de logiciels de dvi, espérant trouver quelque chose que je puisse utiliser. Malheureusement, je n'ai rien trouvé qui puisse m'aider. J'ai donc dû écrire le programme à partir de zéro.

En tout et pour tout, trois types de logiciels dvi existent : utilitaires, pilotes d'imprimantes et prévisualisateurs. Autant que l'auteur le sache, TkDVI d'Anselm Lignau

est le seul programme traitant du dvi qui puisse être utilisé dans le cadre d'un programme interactif.

6 Applications

Nous décrivons ici des applications variées, déjà développées ou en cours. En théorie, des projets possibles seraient un navigateur web avec un support pour les mathématiques ou un éditeur WYSIWYG pour TEX ; nous les espérons à long terme. Ici, les objectifs sont des petits projets, simples, autonomes dans une large mesure, qui sont d'une utilité immédiate et nous font progresser.

6.1 Vitrine d'exposition pour TEX

Une des grandes forces de TEX est son algorithme de coupures de lignes. Comme il est globalement optimisé, un changement à la fin d'un alinéa peut affecter la coupure de la première ligne. Nous connaissons tous cela en théorie mais l'observons rarement en pratique. Dès lors que le démon TEX nous fournit un retour instantané, il est maintenant possible d'écrire une application qui démontre l'algorithme de coupure des lignes de TEX. En d'autres termes, on peut voir le changement de la composition d'un alinéa au gré des changements des paramètres et pourquoi pas du contenu.

D'autres parties de TEX comme les mathématiques et la composition des tables peuvent être ainsi « mises en vitrine ».

6.2 Didacticiel interactif

Les nouveaux venus à TEX ont souvent besoin de beaucoup de retour visuel pour renforcer dans leur esprit le rapport entre, d'une part, les caractères qu'ils tapent et, d'autre part, les mots et formules qui apparaissent sur la page. En utilisant le démon TEX comme moteur de composition, on peut construire un didacticiel interactif pour aider le débutant à apprendre LATEX ou tout ce qu'il voudra savoir des macros de son paquetage favori.

6.3 *Instant Preview*

Ce programme été présenté à EuroTEX 2001 [2]. Voici comment il fonctionne. Supposons que le tampon actif soit en mode de prévisualisation. Alors, à chaque appui sur une touche du clavier, une petite région qui contient la partie visible du tampon est envoyée au démon TEX puis aussitôt affichée dans une fenêtre xdvi. Cela donne *Instant Preview*.

Références bibliographiques

[1] Jonathan FINE: "Active TEX and the DOT Input Syntax". *TUGboat*, Vol. 20, no. 3, p. 248–254. September 1999.

[2] Jonathan FINE: "Instant Preview and the TEX Daemon". In: *EuroTEX 2001*, (p. 49–58). Kerkrade, The Netherlands. September 2001.

[3] Donald Ervin KNUTH: "The Errors of TEX". *Software—Practice and Experience*, Vol. 19, p. 607–685. Reprinted in [4]. 1981.

[4] Donald Ervin KNUTH: *Literate Programming*. No. 27 in Lecture Notes. Center for the Study of Language of Information. 1992.

[5] Donald Ervin KNUTH: *Digital Typography*. No. 78 in Lecture Notes. Center for the Study of Language of Information. 1999.