Spécification et conception d'un éditeur de texte page

H4203 – Cycle R. AUBRY

14 novembre 2010

Table des matières

Première partie

Introduction

Le document suivant présente la conception et la spécification d'un éditeur de texte page glissante. L'éditeur s'adressant à un utilisateur débutant en informatique, très peu de fonctions sont proposées à l'utilisateur afin de limiter le temps d'apprentissage nécessaire. Le texte affiché est brute, *i.e.* sans traitement et sans formatage.

Nous nous concentrerons en priorité sur l'ergonomie, puis sur la portabilité, et enfin sur la maintenabilité et l'efficacité.

Deuxième partie

Spécifications

1 Caractéristiques

1.1 Périphériques requis

Périphériques :
- Un souris deux bouton.
- Un clavier standard.
- Un terminal pouvant afficher au moins 80 caractères par ligne, 50 lignes par page, auquel il faut ajouter les quatre lignes appartenant à l'interface de l'éditeur.

L'éditeur de texte ne gère qu'un seul encodage : l'**ASCII étendu**.

1.2 Zone tampon

Cette zone mémoire contient la dernière sélection effectuée.

2 Comportement de l'interface

Curseur : Le curseur est un rectangle semi-transparent,de la hauteur d'une ligne, de la largeur d'un caractère, et pouvant être déplacé via un clic de la souris. Il ne peut être placé que sur un caractère existant ou en début de ligne.

Ligne d'entête : Elle affiche le nom du fichier, son état, le numéro de la page en édition, le numéro de la ligne et de la colonne où se trouve le curseur, et éventuellement le mode de lecture (lecture seule ou lecture/écriture)..

Un fichier ne peut être que dans 2 états :
- Enregistré : La version affichée à l'écran et la version sur le disque sont les mêmes.
- Modifié : La version affichée à l'écran et la version sur le disque sont différentes.

Zone de saisie : Cette zone, réservée à la saisie du texte, fait :
- 80 caractères de large
- 50 lignes de haut

C'est la dimension d'une page *affichée*.

Si l'utilisateur dépasse :
- 80 caractères : Le nouveau caractère est transféré sur la ligne suivante. Aucun saut de ligne n'est cependant ajouté, seul l'affichage est modifié.
- 50 lignes : L'éditeur insère un caractère de fin de page et place le curseur au début de la page suivante.

Dans cette zone, l'utilisateur peut insérer du texte en tapant les caractères qu'il veut insérer au clavier. Les caractères saisis seront placés avant le curseur. Le caractère de fin de ligne est inséré par la touche « Entrée ».

L'utilisateur peut sélectionner du texte en appuyant sur le bouton de la souris, en déplaçant le curseur jusqu'au dernier caractère de la sélection, puis en relâchant le bouton. Le texte sélectionné est automatiquement mis en zone tampon..

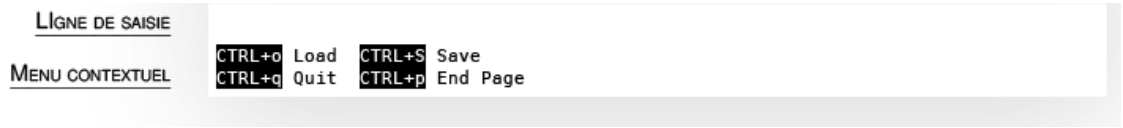
L'utilisateur peut aussi coller au niveau du curseur le texte présent dans la zone tampon (*i.e* le dernier texte sélectionné) en appuyant sur le second bouton de sa souris.

L'utilisateur peut supprimer le caractère courant en appuyant sur la touche standard « Suppr ». Il peut aussi supprimer le texte sélectionné au curseur en sélectionnant le texte, puis en appuyant sur la touche standard « Suppr ».

Page glissante : L'utilisateur a aussi la possibilité de naviguer entre les pages en appuyant sur les flèches directionnelles du clavier :
- « Flèche haut » : Change la page courante avec la page précédente si elle existe.
- « Flèche bas » : Change la page courante avec la page suivante si elle existe.

Dans les deux cas, le curseur est placé en première ligne.

Menu contextuel Constitué de 3 lignes, une *ligne de saisie* et deux lignes d'aide contextuel, il répertorie toutes les commandes accessibles dans le contexte courant sous la forme :



Les différents éléments de l'aide :
- **Charger** : « CTRL + o »
- **Enregistrer** : « CTRL + s »
- **Fin de page** : « CTRL + p »
- **Quitter** : « CTRL + q »

Dans le cas où les commandes de gestion de fichiers sont appelée, l'aide contextuelle affiche (dans le cas d'un système en Français) :

Veuillez saisir le chemin absolu du fichier
Enter : Valider ESC : Annuler

Toutes les saisies relatives aux noms de fichiers seront effectuée dans la *ligne de saisie*.

Gestion des fichiers : L'utilisateur peut charger un fichier existant en appuyant simultanément sur les touches « CTRL + O ». Dans ce cas, le programme demande à l'utilisateur de saisir le chemin absolu du fichier à ouvrir dans la ligne de saisie.

Il peut aussi sauvegarder le texte en cours d'édition. Il est demandé à l'utilisateur de saisir le chemin absolu du nouveau fichier. Si le nom de fichier est le même que celui qui est en cours d'édition, les modifications sont apportées sur le fichier déjà sauvegardé.

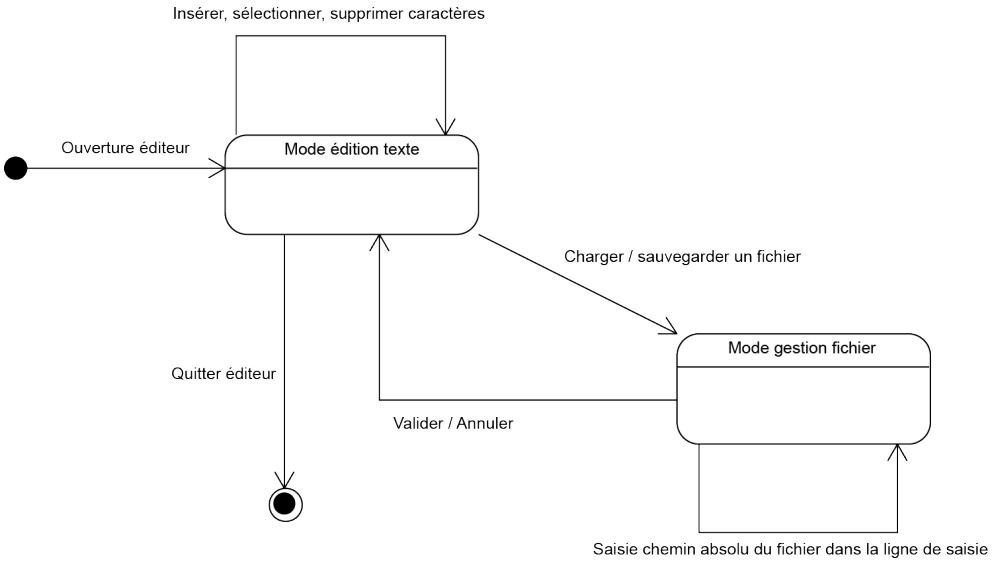
Gestion des langues : La langue de l'éditeur, *i.e.* les messages d'erreurs et les messages du menu contextuel, prennent automatiquement celle du système.

2.1 Lancement

Deux modes de lancement sont disponibles :

1. Sans paramètre : Un fichier vide appelé « Untitled » est ouvert, mais pas enregistré.
2. Avec un paramètre : Le fichier passé en paramètre est chargé et affiché s'il existe.

3 Diagramme comportemental



4 Interface utilisateur

5 Liste des erreurs

Seules les opérations sur les fichiers peuvent générer des erreurs. Les messages dépendent de la langue du système. Ici, pour un système un français :

Le nom de fichier à charger n'existe pas : Le menu contextuel affiche « Fichier inexistant »

Le fichier où aura lieux la sauvegarde est protégé en écriture : Le menu contextuel affiche « Fichier protégé »

6 Bilan

Tout en respectant un cahier des charges réaliste, notre application a le mérite de proposer toutes les fonctions de bases pour manipuler un document source. Il permet, dans un contexte *multilingue*, avec des *requis matériels minimalistes*, quelque soit le *système d'exploitation*, ou la *machine cible* :

- D'effectuer toutes les opérations de fichiers du type : Ouvrir, charger, sauvegarder
- D'insérer, modifier, supprimer du texte
- De naviguer dans un fichier texte, au travers de ses différentes pages

7 Analyse critique du mode opératoire

Dans cette section nous allons comparer le mode opératoire appliqué pour la spécification de l'éditeur de page et les deux autres méthodes proposés pas J.M. PINON et J.F. PETIT.

7.1 Comparaison avec la méthode de J.M. PINON

La méthode présentée par M. PINON est basée sur une analyse très poussée des besoins et l'élaboration des différents diagrammes de cas d'utilisation et de séquences. Cette analyse permet d'avoir une vision exhaustive des besoins de l'utilisateur mais peut devenir assez lourde et relativement chronophage.

En revanche le mode opératoire appliqué pour la réalisation de l'éditeur de texte part d'un étude moins systématique et permet d'établir beaucoup plus rapidement les principales fonctionnalités.

7.2 Comparaison avec la méthode de J.F. PETIT

La méthode de J.F. PETIT est basée sur les objets métiers. Pour extraire ces objets il est nécessaire de réaliser une étude de données. En principe on a un objet métier par fenêtre ce qui est assez rigide et peut dérouter l'utilisateur (le nombre de fenêtres peut devenir très grand). Pourtant, cette méthode a l'avantage d'être assez facile à mettre en œuvre.

Troisième partie

Conception

8 Rappel du contexte

La suite de ce document présente la démarche de conception entreprise dans le cadre de ce projet. Nous souhaitons proposer un éditeur de textes page capable de manipuler efficacement un fichier texte séquentiel de longueur variable.

L'architecture retenue à l'issue de cette phase de conception doit permettre d'offrir un outil ergonomique (aspect essentiellement traité dans le cadre de la spécification), portable sur différents environements et différentes architectures, maintenable et efficace. Nous devons garder en mémoire que cet outil doit pouvoir être déployé dans un environnement où les ressources sont limitées.

Nous analyserons et critiquerons trois solutions de gestion des données en mémoire centrale et mémoire secondaire selon les critères de portabilité, maintenabilité et efficacité. Nous retiendrons celle jugée la plus pertinente et nous la décrirons en détail.

9 Choix d'une solution

9.1 Présentation des alternatives

9.1.1 Points commun des trois solutions

À l'ouverture d'un fichier, son contenu est copié dans un fichier temporaire — éventuellement en le structurant de manière adaptée à l'éditeur. Le fichier est indexé de manière à pouvoir accéder aux lignes directement.

Un tampon de taille limitée est créé en mémoire centrale. Lorsque l'utilisateur demande à accéder à des lignes qui n'y sont pas encore chargées, les lignes les plus anciennement accédées sont remplacées par celles qui viennent d'être demandées.

Le tampon est modifié directement au fur et à mesure de la frappe de l'utilisateur, et la ligne modifiée est transférée dans le fichier temporaire dès que l'utilisateur passe à une nouvelle ligne ou demande une sauvegarde du fichier. Le fichier source, quant à lui, est mis à jour sur demande de l'utilisateur (sauvegarde du fichier).

9.1.2 Solution 1 : structure consécutive

Toute la mémoire est allouée dans des tableaux, d'un seul bloc.

Le fichier temporaire est lui aussi placé dans une structure consécutive dans un zone contigüe de la mémoire centrale.

La solution est donc homogène : le même mécanisme est mis en place entre la mémoire centrale et la mémoire secondaire. La compréhension du système est alors simplifiée.

L'accès est direct : chaque caractère du texte peut être accédé en temps constant ($O(n)$). De fait, si l'on a besoin uniquement de remplacer des caractères par d'autres caractères, cette solution est efficace.

De la même manière, si l'on a besoin de rajouter des caractères à la fin du texte, la solution est là encore très efficace, dans les limites de la mémoire disponible sur le système.

De plus, cette solution présente l'avantage de ne pas devoir réserver de la mémoire pour le chaînage, celui-ci n'étant pas présent dans cette solution. Par contre, nous devons impérativement réserver un *buffer* de taille importante dès le début, pour éviter la réallocation intempestive. Nous pourrions envisager d'utiliser la technique bien connue de la réallocation par pas de n^2, c'est à dire réserver la mémoire selon un schéma quadratique, et donc réserver de plus en plus de mémoire à chaque réallocation.

Par contre, si l'on a besoin d'insérer des caractères au milieu du texte, la solution est très couteuse. En effet chaque insertion ou suppression de caractères provoque des opérations en $O(n)$ (où n est le nombre de caractères entre le point de fin d'insertion ou de suppression et la fin de la mémoire allouée (le *buffer*). Comme il s'agit d'une opération assez classique et fréquente, de nombreuses opération sont à prévoir. Ces opération sont du type `memmove()`, qui permet de déplacer des données dans un *buffer*, et qui a une complexité de n.

9.1.3 Solution 2 : structure mixte

La seconde solution proposée est une variation de la première. La structure de données en mémoire secondaire reste identique : nous conservons le fichier sous la forme d'une structure de caractères contigües. Cependant, le bloc texte est fragmenté en sous-blocs de taille donnée en mémoire centrale et stockée dans des éléments d'une liste doublement chaînée. Les éléments sont chaînés dans l'ordre des données du fichier.

La structure d'un élément de la liste contiendra les données suivantes :

@précédant Adresse de l'élément précédant dans la liste,

@suivant Adresse de l'élément suivant dans la liste,

lgUtil Longueur du bloc utilisée données,

lgMax Longueur totale du bloc de données,

info Bloc de données (contient le fragment de texte stocké dans l'élément).

Le bloc de données peut éventuellement être plus large que la chaîne de caractères qu'il contient. En effet, lors de l'édition du document, de nouveaux blocs peuvent être alloués mais pas intégralement remplis par la saisie de l'utilisateur. Il est donc nécessaire de retenir deux informations sur ce bloc : $lgUtil$, la longueur effectivement utilisée, qui correspond à la taille de la chaine stockée (marqueur de chaîne inclus), et $lgMax$, la taille du bloc $info$.

Cette seconde solution est moins efficace que la première lors du chargement et de la sauvegarde du fichier. Les structures manipulées en mémoire centrale et mémoire secondaire ne seront plus homogènes, il sera donc nécessaire de proposer des procédures permettant de convertir le bloc de texte d'un format de structure à l'autre. Les opérations de chargement et sauvegarde du fichier seront plus coûteuses du fait de ces opérations de traduction, mais aussi des opérations d'allocation de mémoire proportionnelles à la taille du fichier.

Par ailleurs, cette solution est plus coûteuse en espace pour la mémoire centrale, puisqu'il est nécessaire de prévoir quelques octets correspondant à la taille des pointeurs manipulés.

En contrepartie, cette solution offre de biens meilleurs temps d'accès en lecture et écriture, puisqu'il est possible de naviguer d'un élément de la chaîne à un autre sans lire l'ensemble du bloc. La

complexité algorithmique d'une telle opération sera en $O(n/m)$, où n est la longueur maximale d'un bloc et m le nombre d'éléments de la chaîne. L'insertion de texte nécessite de se placer à la fin du bloc de données et d'ajouter le caractère saisi. Si la longueur de la chaîne atteint la longueur maximale, il sera nécessaire d'allouer un nouvel élément et de l'insérer dans la chaîne à la suite de l'élément courant.

Une double liste chaînée est une structure de données classique qui n'est pas difficile à implémenter et à manipuler. L'impact d'une telle solution sur la maintenabilité est très faible.

9.1.4 Solution 3 : structure chaînée

Cette solution étend la notion de chaînage à la mémoire secondaire.

Le fichier temporaire n'a plus une structure séquentielle, mais doublement chaînée : chaque ligne est stockée dans une « enveloppe » contenant non seulement les caractères formant cette ligne, mais aussi des adresses virtuelles permettant de retrouver, en accès direct, les lignes suivante et précédente.

Puisque le fait de rajouter cette « enveloppe » va de toute manière « casser » la structure initiale du fichier texte, on peut également se permettre de rajouter d'autres informations : par exemple, la longueur utile de cette ligne, et la longueur maximale. Ainsi lors d'une suppression de caractères dans une ligne, aucun déplacement de données dans le fichier n'est nécessaire : on se contente de modifier les caractères et de réduire la longueur utile.

L'insertion et la suppression de ligne sont alors très simple :
- Une insertion se fait en ajoutant l'« enveloppe » de la ligne dans un espace libre du fichier temporaire, éventuellement à la fin, et en modifiant les chaînages des lignes précédente et suivante.
- Une suppression se fait simplement en modifiant les chaînages des lignes précédente et suivante.

La mémoire centrale est gérée de la même manière ; les blocs sont stockés tels quels dans un *buffer* (avec l'adresse virtuelle).

Lorsqu'on affiche une page, on part de la première ligne. Le bloc correspondant est retrouvé grâce à un système d'indexation sur le fichier temporaire, et il est copié dans le *buffer*. On affiche les données de la ligne, puis on lit l'adresse virtuelle de la ligne suivante : si elle correspond à un bloc non chargé en mémoire centrale, on le charge. Puis on affiche les données, on lit l'adresse de la ligne suivante, ...

9.2 Comparaison des alternatives

Les critères d'ergonomie et de portabilité ne sont pas pertinents pour la comparaison des alternatives. En effet, une bonne ergonomie, au niveau de la conception, sera traduite par une bonne efficacité, afin d'éviter à l'utilisateur des attentes inutiles. De plus, nous n'utiliserons pas d'assembleur dans les différentes solutions, se reposant sur le compilateur C++ pour assurer cette portabilité. Le système sera donc portable tant que la plateforme dispose d'un compilateur C++.

Nous avons choisis de pondérer ces critères, pour prendre en compte leur importance respectives.

Critère	Pondération	Solution 1	Solution 2	Solution 3
Maintenabilité	20	++ (34)	++ (35)	+++ (56)
Décomposable en couches	8	+	++	+++
Homogénéité	5	+++	+	+++
Couplage entre couches	5	+	++	+++
Facilité d'implémentation	2	+++	++	+
Efficacité	15	++ (27)	++ (27)	+++ (41)
Complexité en insertion	3	+	++	+++
Complexité en suppression	3	+	++	+++
Complexité au changement de ligne	3	+	+	+++
Complexité de changement de page	3	+++	++	+++
Économie mémoire	1	+++	++	+
Complexité de chargement du fichier temporaire	1	+++	++	+++
Complexité de la sauvegarde	1	+++	++	+
Total	35	61	60	98

TABLE 1 – Table comparative des avantages et inconvénients de chaque solution

10 Détail de la solution choisie

10.1 Structuration d'une ligne en mémoire secondaire et en mémoire centrale

Le ficher temporaire est chaîné grâce à des adresses virtuelles, c'est à dire des chiffres permettant de retrouver une ligne dans le fichier temporaire.

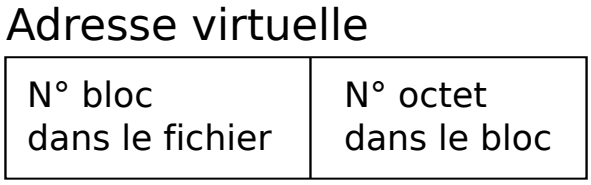
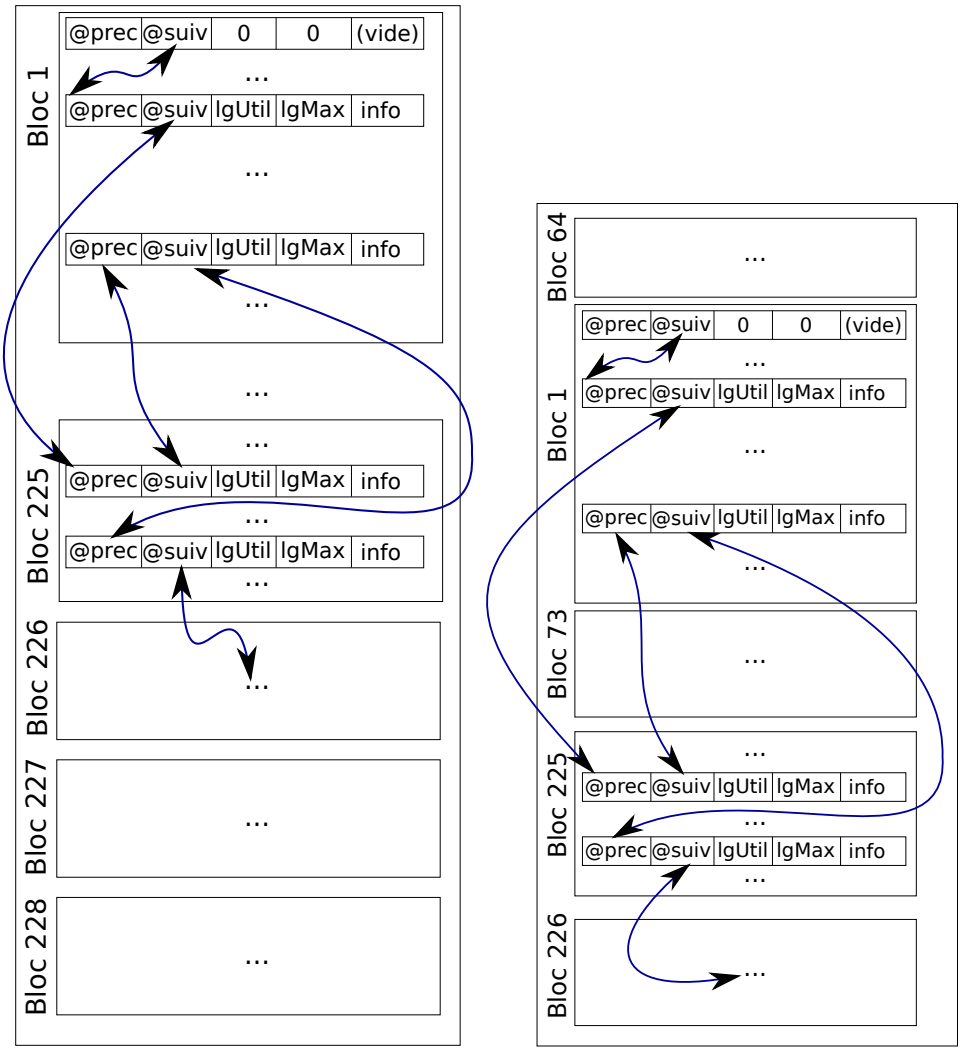


FIGURE 1 – Structure d'une adresse virtuelle

Ainsi chaque ligne fait référence à son successeur et son prédécesseur. On stocke également la longueur de l'information (« longueur utile ») et le nombre d'octets disponibles (« longueur max », supérieure ou égale à la longueur utile) dans l'enregistrement correspondant à une ligne.

La même structure est conservée en mémoire centrale, mais les blocs ne sont pas forcément dans le même ordre [1]. Cela permet de retrouver facilement sur le disque une ligne devant être chargée par la suite.

1. Cf. page 14 : « Gestion des blocs sur la mémoire secondaire pour faire de l'accès direct et gestion du texte en mémoire centrale »



(a) Structure du fichier temporaire (b) Structure du tampon en mémoire centrale

FIGURE 2 – Structure des données de la solution 3 (exemple)

- Lors d'une insertion, l'algorithme classique d'insertion d'un élément dans une liste chaînée est utilisé. Le nouvel enregistrement est placé au premier espace libre du fichier temporaire quoi soit assez grand pour l'accueillir[2], et les adresses concernées (précédente, suivante, suivante de la précédente, précédente de la suivante) sont mises à jour.
- Lors d'une modification entraînant un dépassement de la taille maximum de la ligne, on la déplace dans le premier espace libre assez grand pour l'accueillir, et on met à jour les adresses des lignes contigües.
- Lors d'une suppression, on met simplement à jour les adresses des lignes contigües afin d'enlever la ligne supprimée du chaînage.

On remarquera qu'une ligne « zéro » a été ajoutée, ceci pour permettre de généraliser les opérations d'insertion et de suppression de ligne. De même, une ligne de « fin » est présente (non représentée sur les schémas). Aucune de ces deux lignes n'est affichée à l'écran.

10.2 Mécanisme de stockage en mémoire secondaire permettant l'accès direct

L'accès direct dans le fichier temporaire est possible grâce à la table de *mapping*, qui donne, pour une partie des lignes, l'adresse virtuelle de celles-ci.

2. Cf. page 15 : « Gestion de l'espace libre »

N° ligne	Adresse virtuelle
10	bloc 226, octet 156
18	bloc 226, octet 212
25	bloc 227, octet 61
...	
110	bloc 54, octet 451
120	bloc 54, octet 564

TABLE 2 – Structure de la table de *mapping* de la solution 3 (exemple pour $L = 10$)

À la construction du fichier, on mémorise dans cette table l'adresse virtuelle d'une ligne toutes les L lignes (L à définir). La table est indexée par le numéro de ligne.

Lors de l'ajout d'une ligne, la table est mise à jour :
- Les enregistrements concernant les lignes suivantes voient leur index incrémenté (le numéro de ligne est mis à jour).
- Si la ligne précédant celle insérée dans la table et la ligne suivante ont désormais une différence d'index supérieure à L, une ligne est choisie entre les deux pour être insérée dans l'index.

Lors de la modification d'une ligne indexée entraînant un déplacement de celle-ci, son adresse dans la table de *mapping* est mise à jour.

Lors de la suppression d'une ligne indexée, elle est supprimée de la table.

Lorsque l'on désire accéder directement à une certaine ligne, il suffit de trouver la ligne la plus proche qui soit indexée dans la table, puis de suivre le chaînage. Grâce aux opérations réalisées à l'ajout d'une ligne, on s'assure de borner (par 0 et L) le nombre de chaînages à parcourir avant de trouver la ligne désirée. L'accès n'est donc pas réellement *direct*, mais il en a les propriétés (la complexité algorithmique constante).

Cette solution permet d'éviter de stocker une entrée dans la table pour chaque ligne, ce qui pourrait se révéler coûteux en mémoire dans le cas d'un très grand fichier.

Le lecteur attentif aura noté que la table comporte un risque de dégénérescence, si toutes les lignes non indexées sont supprimées : alors toutes les lignes restantes seront indexées. C'est pourquoi on fixera un seuil $Lmin$: lorsque, à la suite d'une suppression, un index se retrouvera compris entre deux autres dont l'écart d'index est inférieur à $Lmin$, il sera « désindexé ».

10.3 Gestion des blocs sur la mémoire secondaire pour faire de l'accès direct et gestion du texte en mémoire centrale

La structure des blocs en mémoire centrale et en mémoire secondaire est homogène, il est ainsi possible de charger facilement un bloc en mémoire secondaire qui ne serait pas encore disponible en mémoire centrale. Nous utilisons le mécanisme d'accès direct à la mémoire secondaire décrit précédemment pour lire un bloc à copier en mémoire centrale.

Il nous manque cependant un moyen de faire la correspondance entre un bloc en mémoire secondaire et un bloc en mémoire centrale, nous allons donc mettre en place une table de correspondance (qu'on appellera *Table de mapping MC-MS*) qui contiendra, pour chaque entrée, l'adresse virtuelle du bloc en mémoire centrale et l'adresse virtuelle du bloc en mémoire secondaire. Ce tableau est trié par numéro de bloc.

N° Adresse Virtuelle MC	Adresse Virtuelle MS
bloc 1, @150	bloc 1, @40
bloc 2, @190	bloc 2, @80
bloc 3, @230	bloc 3, @120
...	

TABLE 3 – Structure de la table de *mapping MC-MS* de la solution 3

Différents cas de figures font intervenir l'utilisation de cette table :
- Le chargement d'un bloc en mémoire centrale d'après la mémoire secondaire
- Le chargement d'un bloc en mémoire secondaire d'après la mémoire centrale
- La mise à jour d'un bloc de la mémoire centrale vers la mémoire secondaire

Lorsque nous souhaitons *charger un bloc en mémoire centrale d'après la mémoire secondaire*, le processus est similaire à la création d'une nouveau bloc en mémoire centrale lorsque l'utilisateur saisit du texte : un nouveau bloc est créé à un emplacement libre de la mémoire centrale et sera chainé à la ligne « fin ». Ensuite, les informations du bloc en mémoire secondaire ($lgUtil$, $lgMax$ et $info$) seront copiées dans le bloc nouvellement créé en mémoire centrale. Il nous faut alors mettre à jour la table de Mapping MC-MS en ajoutant une entrée au tableau contenant l'adresse du bloc créé et celle du bloc original en mémoire secondaire.

Pour éviter de surcharger la mémoire centrale, un nombre critique de blocs sera conservé. Si ce nombre est atteint, alors le premier bloc de la liste (suivant la ligne « zéro ») est supprimé de la mémoire centrale avant la création du nouveau bloc. Cette valeur impacte le nombre de ligne que l'utilisateur peut visualiser sur un même écran et indique la taille de la table de Mapping MC-MS.

La suppression d'un bloc en mémoire centrale implique que celui-ci ai été recopié en mémoire secondaire pour le que les données du fichier soient conservées. Deux cas de figure sont alors à envisager : soit le bloc existe déjà en mémoire secondaire (mais n'est potentiellement plus à jour par rapport à la version en mémoire centrale), soit le bloc n'existe pas encore en mémoire secondaire et il faudra le créer.

Dans la première situation (correspondant au cas *mise à jour d'un bloc de la mémoire centrale vers la mémoire secondaire*), nous utilisons la table de Mapping MC-MS pour obtenir l'adresse du bloc en mémoire secondaire et mettre à jour son contenu ($lgUtil$, $lgMax$ et $info$).

Dans le second cas (*chargement d'un bloc en mémoire secondaire d'après la mémoire centrale*), il faudra créer un nouveau bloc en mémoire secondaire à un emplacement disponible (par exemple en queue du fichier) et chaîner le nouveau bloc aux blocs contenant la ligne précédente et la ligne suivante en mémoire secondaire. La table de Mapping MC-MS est ensuite mise à jour : on ajoute l'entrée correspondant au bloc qui vient d'être créé en mémoire secondaire.

10.4 Gestion de l'espace libre

Lorsque beaucoup de modifications ont lieu sur le document actuellement édité, il peut se produire un phénomène de fragmentation dans la structure de donnée.

Le phénomène de fragmentation se produit lorsque la taille de l'espace libre dans les maillons est trop faible pour effectuer une nouvelle insertion dans ce même maillon (voir page 12 pour le procédé d'insertion). L'espace libre est alors « perdu », puisqu'aucun caractère n'y sera inséré.

Ce phénomène rend de moins en moins efficace la solution technique, par la même la vitesse du logiciel, et donc finalement l'ergonomie. En effet, plus la fragmentation est importante, plus le

nombre de blocs augmente. De plus, le nombre d'éléments dans la liste chainée augmente de la même manière, ce qui augmente le temps de recherche dans la structure de donnée. Enfin, un nombre plus important de maillons et de bloc signifie aussi une utilisation mémoire encore plus importante, puisque chaque maillon occupe des information annexes (pointeurs, informations sur le nombre de caractères utilisés et libres...).

Il semble donc absolument nécessaire de remédier à ce problème.

Un procédé de défragmentation pourra être appelé, soit manuellement, soit à la sauvegarde, et reconstruira les structures de données de manière optimale, c'est à dire de façon à minimiser l'espace non utilisé dans les structures de données. Une solution pourrait consister à repartir d'un fichier d'un seul bloc et de reconstruire le double chainage et les adresses virtuelles. La sauvegarde semble alors être un bon moment pour effectuer cette opération, puisque la « mise à plat » de la structure de donnée sera déjà effectuée pour l'enregistrement sur disque.

11 Proposition d'architecture

L'architecture choisie sera organisée en couches, afin de découpler le code, et de permettre une meilleur maintenabilité. On pourrait même imaginer insérer d'autres couche entre celles présentes dans le tableau ci-dessous, afin de, par exemple, ajouter facilement un mécanisme d'historique, qui semble intéressant pour l'édition de code source. Elle se placerait entre la couche interaction utilisateur et la couche page, sans nécessité pour autant de toucher à ces deux modules (sauf, bien sur, pour ajouter les nouvelles fonctionnalités dans l'interface homme-machine).

Nom de la couche	Description succincte
Couche d'interaction utilisateur	Prend en charge l'ensemble de l'interaction utilisateur : entrée de commande, et retour visuel.
Couche page	Prend en charge le mécanisme de découpage en pages du texte chargé.
Couche ligne	Prend en charge le mécanisme de découpage en ligne d'une page.
Couche adressage	Implémente le mécanisme d'adressage virtuel pour l'accès direct présenté dans les parties précédentes
Couche physique	Prend en charge les appels système pour interagir avec le disque.

Il sera possible de remplacer une couche par une autre, afin d'adapter l'éditeur à des cas très précis, sans devoir refaire entièrement son développement. Par exemple, le logiciel pourra être adapté pour des personnes ne pouvant pas utiliser un clavier, simplement en changeant la couche d'interaction utilisateur par un module de reconnaissance vocale.

12 Bilan

La solution que nous avons choisi repose sur une architecture sélectionnée parmi trois propositions, selon les critères définis par les exigences du cahier des charges.

Les trois architectures proposées répondaient aux exigences fonctionnelles, notre étude s'est donc naturellement portée sur l'évaluation de la meilleure solution technique répondant aux exigences de portabilité, maintenabilité et efficacité.

La solution retenue était de loin meilleure que les deux autres pour l'ensemble des critères. Elle repose sur des structures algorithmiques bien connues et couramment utilisées, ce qui rend son architecture abordable.

Cette solution offre par ailleurs à l'utilisateur une bonne réactivité globale de l'application pour chacune de ses actions, au prix cependant d'un coût certain en mémoire : les données permettant d'indexer le texte manipulé sont conservées et permettent d'accélérer les traitements grâce, notamment, à des accès directs.

Cette solution offre un ultime avantage : elle est paramétrable. En effet, il est possible de faire varier le nombre de blocs conservés en mémoire centrale, la taille d'un bloc et le pas d'indexation des lignes de texte pour choisir un bon compromis entre l'occupation de la mémoire centrale et la réactivité de l'application.