Édition distribuée de partitions musicales multi-polices

Nabil Bouzaïene & Emmanuel Saint-James

[bouzaien,esj]@liafa.jussieu.fr
Laboratoire d'Informatique Algorithmique Fondements et Applications
4 place Jussieu
75252 Paris cedex 5

1. Introduction

L'édition électronique de partitions musicales vise pour le moment à imiter plus ou moins bien des documents imprimés. Pourtant, la numérisation de leur support ouvre des domaines inaccessibles à la gravure traditionnelle. La mise au point de polices de caractères représentatives de l'esthétique musicale sous-jacente en fournit déjà un exemple: théoriquement faisable avec les moyens traditionnels, elle n'a pu être réalisée qu'avec l'abaissement des coûts autorisé par l'informatique [1].

Un autre domaine nouveau, rendu aisé par l'essor spectaculaire du Word Wild Web, est l'édition d'une partition par plusieurs intervenants simultanément. Un avantage quantitatif évident est la rapidité de production permise par le travail simultané, particulièrement manifeste dans les œuvres sollicitant un grand effectif orchestral.
Un avantage qualitatif est que chaque portée de la partition peut être saisie par un musicien connaissant bien l'instrument, donc particulièrement compétent pour éviter les fautes de copies et lever les ambiguïtés.

Mais surtout, cette mise en réseau permet à un groupe de musiciens de préparer l'interprétation d'une œuvre en annotant identiquement leurs partitions, sans avoir nécessairement à se rencontrer physiquement: un trio à cordes multi-national peut par exemple décider à travers le Web des diminuendo & crescendo, des successions tiré/poussé de l'archet etc. avant de se retrouver effectivement pour un concert, voire jamais pour un enregistrement multi-pistes en différé.

Le présent article propose donc un tel outil d'édition. Il se présente comme une applet Java fonctionnant sous un navigateur Web. À la base, c'est un éditeur de partitions multi-polices (grâce à un interprète Embedded PostScript intégré, écrit en Java) pouvant être utilisé en solitaire, mais gérant des requêtes d'édition partagée lorsque d'autres utilisateurs surviennent pour travailler sur la même partition.

La section suivante montre comment résoudre les limitations des navigateurs en matière de police, ce qui nécessitera une étude assez fine des différences entre Java et PostScript. La section d'après détaille les problèmes d'accès concurrents à une partition. On conclura sur les perspectives d'avenir.

2. Dessin de caractères

2.1. Le problème des polices dans une page WEB

Un navigateur WEB a un nombre de polices de caractères limité (6 pour Netscape 3.0 par exemple). Pour visualiser une simple chaîne de caractères dans une police non prédéfinie, il faut en remplacer une sur la machine cliente.
De plus, pour qu'un utilisateur puisse utiliser cette nouvelle police, il faudrait qu'il la télécharge et l'installe au préalable, ce qui ferait perdre à l'applet sa portabilité. Il faut donc que la police soit embarquée dans l'applet, pour qu'au moment du téléchargement de celle-ci la police soit déjà prête. Cela permettra en plus de changer de police en cours d'exécution sans être obligé de passer par une phase de réinstallation.
En réalité, il n'est pas possible d'utiliser une police de caractères qui se trouve sur la machine hôte, il faut contourner le problème en lisant ses polices en tant que fichiers texte.
L'ensemble des polices de caractères est donc sur la machine qui héberge l'applet, sous un format PostScript, et à partir de la machine cliente, l'applet accède à ces fichiers en fonction de la demande de l'utilisateur final.

2.2. Un mini-interprète EPS en Java

Pour permettre des agrandissements (zoom) de la fenêtre de travail, il faut résoudre le problème des marches d'escalier qui apparaissent quand un simple affichage à grande échelle est réalisé.

> Pour cela, il faut redessiner le caractère à la bonne échelle, donc avoir une description de chaque caractère en terme de courbes et non en terme de points (selon l'opposition classique Vectorielle/bitmap) [2].

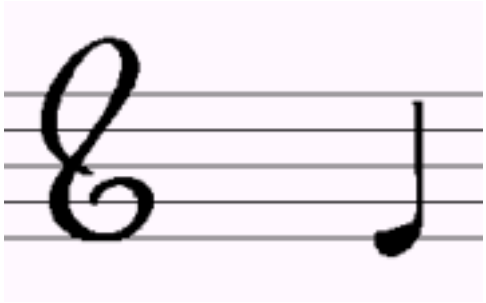

Fig 1 : Les marches d'escaliers

Le logiciel de création de polices de caractères, "Fontographer" permet d'exporter les polices en EPS, en fait, il est possible d'avoir un fichier contenant une liste de fonctions PostScript dessinant tous les caractères à une certaine échelle[1], voici par exemple la fonction du caractère '0' (de la police Sonata):

```
%%[Char name=], Char number, x offset, y offset, x width, y width,
unicode value
%%Note:zero=48 13.4445 719 45 0 48
13.4445 719 m
s
13.6244 719.218 m
13.6244 719.219 l
13.7523 719.218 13.8043 719.083 13.8043 719 c
13.8043 718.918 13.7523 718.783 13.6244 718.783 c
13.4965 718.783 13.4445 718.918 13.4445 719 c
```

[1] Dans la sortie EPS de Fontographer, il y a une petite erreur, il utilise une fonction PostScript 'n' qu'il ne définit pas auparavant, il faut juste remplacer tous les 'n' par 'newpath' par exemple .

```
13.4445 719.083 13.4965 719.219 13.6244 719.218 c
s
13.6244 719.188 m
13.5584 719.188 13.5484 719.049 13.5484 719 c
13.5484 718.952 13.5544 718.815 13.6244 718.815 c
13.6944 718.815 13.7004 718.952 13.7004 719 c
13.7004 719.049 13.6904 719.188 13.6244 719.188 c
s
13.8043 719 m
s
```

Les lettres m, c, l et s sont respectivement les fonction moveto, lineto, curveto et stroke du PostScript.
Un caractère est constitué d'un ensemble de parcours disjoints (comme l'exemple ci-dessus), le caractère de notre exemple est en fait défini à l'aide de deux parcours :

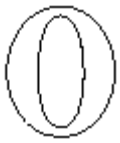

Fig 2 : Le zéro est formé par deux parcours disjoints

Le programme réalisé, part du fichier EPS contenant la description des contours des caractères puis en extrait les valeurs numériques et les différentes fonctions (moveto, curveto, etc...). Chaque caractère est donc représenté sous la forme d'un tableau, contenant l'ensemble des fonctions et des valeurs numériques nécessaires à son dessin.
Les fonctions "MoveTo" et "LineTo" existent dans les classes de l'AWT de Java, par contre le fonction CurveTo n'existait pas encore (elle vient de faire son apparition dans la grosse API Java2D). il a donc fallu programmer un petit algorithme qui décompose la courbe de Bezier en segments de petite
longueur.

2.3. Le remplissage de Java n'est pas celui de Post-Script

Le problème à résoudre est donc de remplir les surfaces délimités par ces parcours, puisqu'il faut savoir à chaque fois si le parcours est à colorer ou pas, et si oui comment colorer juste l'espace entre les deux parcours.
En PostScript le problème ne se pose pas pour deux raisons : d'un côté les différents algorithmes de remplissage se basent sur la notion de parcours, alors que l'algorithme de remplissage de Java, utilise la notion de forme.
De l'autre, le remplissage des parcours en PostScript se comporte d'une façon différente s'il traite un caractère appartenant à un police, ou s'il traite une forme quelconque [3].

La solution apportée est la suivante : au moment du dessin final à une échelle donnée, le caractère est transformé en un ensemble de polygones correspondant chacun à un parcours différent. À ce niveau, et puisque les parcours sont complètement disjoints, un calcul de la "profondeur" du parcours est réalisé de la façon suivante :

- Un polygone qui n'est contenu dans aucun autre à la profondeur 0
- Un polygone contenu dans un autre à la profondeur 1
- Un polygone contenu dans deux autres à la profondeur 2
- Etc…

Ensuite, tous les polygones de profondeurs paires sont remplis en noir, et tous les polygones de profondeurs impaires en couleur de fond. Enfin il faut dessiner les polygones dans l'ordre de profondeur croissante. Cet algorithme donne les mêmes résultats que l'algorithme pair-impair de PostScript Pour des parcours convexes complètement disjoints [3].

Fig 3 : Les parcours en pointillés sont coloriés en couleur de fond

3. Edition partagée

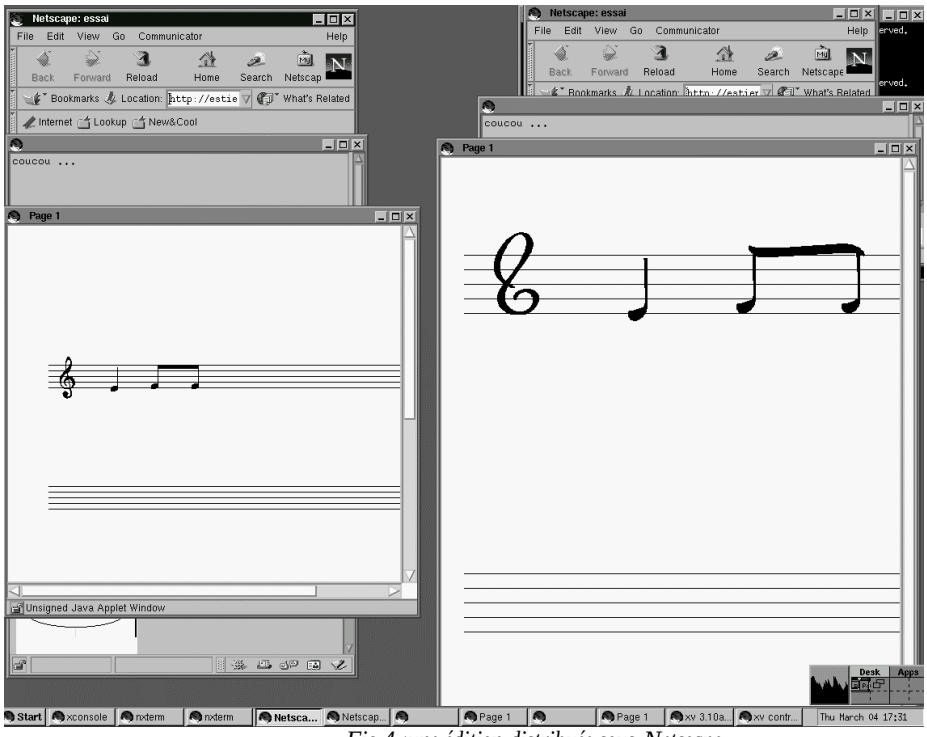

Fig 4 :une édition distribuée sous Netscape

Pour permettre une édition partagée dans de bonnes conditions et pour éviter tous les problèmes que pourrait provoquer une déconnexion du réseau, il faut que chaque client ait sa propre structure de données et qu'il informe les autres des modifications qu'il opère dessus.
Par ailleurs, pour que justement les clients puissent visualiser la même page à des échelles différentes et avec des polices différentes, il faut qu'il y ait deux systèmes de coordonnées : des coordonnées absolues, communes à tous les clients, et des coordonnées propres au contexte d'affichage de chaque

client. Les clients dialoguent donc, en utilisant les coordonnées absolues, que chacun reconvertit en fonction de son contexte d'affichage.

3.1. L'asynchronisme des chargements

Un serveur tourne en permanence sur la machine mère. Au démarrage l'applet "client" essaie de s'y connecter, si elle y arrive elle l'informera de toutes les opérations d'édition qu'elle opère sur la page courante. Le serveur transmet alors l'information à tous les autres clients connectés au même moment. Le résultat est qu'un client donné n'est informé que des opérations qui ont lieu à partir du moment de sa connexion.
Cette méthode présente néanmoins un problème illustré par l'exemple ci-dessous :
Soient deux Clients A et B travaillant sur la même partition :
Le client A envoie le message : "suppression de la zone 50 50 100 100" qui veut dire qu'il efface tous les composants qui sont situés dans cette région de la page. Au "même" moment le client B, rajoute une note justement dans cette zone avant de recevoir le message d'effacement du client A. Dans ce cas il peut exister un conflit dont voici un bilan chronologique:
- Le client A efface la dite zone sur sa page en informe le serveur et la voit effectivement s'effacer.
- Le client B rajoute une note, en informe le serveur, et la voit apparaître sur sa page.
- Pendant ce temps le message de A arrive au serveur, il en informe B qui efface donc la note de sa page.
- Et enfin le message de B arrive au serveur qui en informe A qui rajoute donc la fameuse note.
Résultat : A et B n'ont pas la même partition !

La deuxième solution qui empêche ce comportement, consiste à faire en sorte que le serveur informe tous les clients (y compris l'envoyeur) de tous les événements , et que chaque client attende le retour du serveur pour effectivement procéder au changement , notre exemple devient :
- Le client A demande l'effacement de la zone mais ne fait rien pour l'instant.
- Le client B demande l'ajout d'une note et attend.
- Le message de A arrive au serveur qui en informe A et B, qui effacent la zone sur les deux pages.
- Le message de B arrive au serveur qui en informe A et B, qui rajoutent tous les deux la note sur la page.
Résultat: A et B ont la même partition.
Cette solution est satisfaisante dans le cas où il y a effectivement plusieurs clients qui interviennent au même temps, mais fait perdre à l'application une fluidité d'exécution même dans le cas où il y a un seul client.

Finalement, une troisième solution repose sur deux modes d'exécution un "local" et l'autre "en réseau". Quand un client demande une nouvelle connexion au serveur, celui-ci teste s'il existe d'autres clients déjà connectés.
S'il n'y en a pas, alors il informe le client qu'il est en mode "local". Celui-ci n'avertit plus le serveur de ses changements mais reste, cependant, à son écoute.
Par contre, si le serveur trouve un seul autre client déjà connecté, alors il informe les deux clients (l'ancien et le nouveau venu) qu'ils sont en mode "réseau", et qu'il doivent informer le serveur de tous les changements.
De plus, il demande à l'ancien de mettre à jour la page du nouveau, en lui transmettant l'état (toutes les notes) de la page courante.
Enfin si le serveur trouve plusieurs autres clients déjà connectés, il informe le nouveau client qu'il est en mode "réseau" puis demande au plus ancien client de mettre à jour le nouveau.
Voici un organigramme qui résume l'action du serveur au moment d'une connexion :

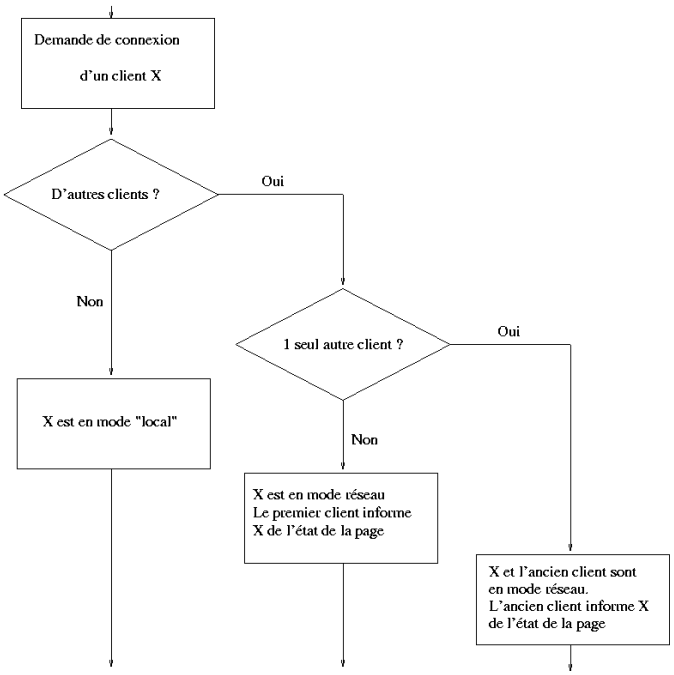

Fig 5 : Activité du serveur au moment d'une nouvelle connexion

3.2. fenêtre textuelle

Pour permettre un dialogue entre les différents intervenants sur une même partition, une fenêtre textuelle est ajoutée. Il est donc possible de s'envoyer des messages textuels comme dans une session de "chat", puis de se remettre dans la page de travail pour agir sur la partition.
Le serveur différencie les messages textuels des messages de travail grâce à un mot clef ajouté au début des messages textuels.

4. Conclusion

Les problèmes de sécurité bien connus sur le Web limitent la version actuelle de l'éditeur: une applet java ne peut ouvrir un fichier sur la machine hôte, il n'est donc pas possible de retravailler une partition saisie préalablement. Partant, la sauvegarde est minimale: c'est un simple mémorisation des notes, sous le format abc [4], évacuant volontairement toute indication graphique. Libre à l'utilisateur ensuite d'imprimer le résultat par un éditeur type abc2ps, ou de faire des copies d'écran pendant sa session.
L'adoption d'un format plus riche, type NIFF, est à l'étude, ainsi que l'élaboration de droits d'accès sur le serveur de l'éditeur afin de l'étendre à un atelier de composition de partitions.

5. Bibliographie

[1] Bouzaïene, Le Gall & Saint-James, *Une bibliothèque informatique pour la notation musicale baroque*, Lecture Notes in Computer Science 1375, Springer 1998

[2] Jacques André & Irène Vatton, *Dynamic optical scaling and variable-sized characters*, Electronic Publishing vol 7(4), Décembre 1994

[3] *Manuel de référence du langage PostScript*, Deuxième édition, Addison-Wesley 1992

[4] Chris Walshaw, *The abc musical notation language*, http://www.gre.ac.uk/c.walshaw/abc/

[5] J.Gosling, B.Joy & G.Steele,*The Java language specification*, Addison-Wesley 1996