HOLO-EDIT ONLINE : CONSTRUCTION D'UNE APPLICATION JAVA SUR INTERNET

Laurent Pottier, Léopold Frey
Gmem - Centre National de Création Musicale
15-17 rue de Cassis
13008 Marseille
Tél : 04.96.20.60.10
Fax : 04.96.20.60.19
dvlpt@gmem.fr

RÉSUMÉ

Ce texte présente les évolutions apportées au logiciel « Holo-Edit », inclus dans la suite logicielle pour la spatialisation éditée par le Gmem : Holophon[2][5]. Ce logiciel permet de programmer des trajectoires sonores dans un espace tridimensionnel. Ces trajectoires peuvent être utilisées par la suite dans Holo-Spat, spatialisateur temps-réel modulable réalisé sous Max/MSP (communication par bus inter-application ou par fichier), ou dans tout logiciel sachant utiliser des fichiers au format Midifile.

Les principales évolutions concernent son portage vers les plateformes les plus courantes (systèmes Apple, Microsoft et Unix) grâce à Java mais également sa disponibilité en ligne. Ce document présente les principaux problèmes rencontrés pour cette mise en ligne, ainsi que les solutions adoptées pour les résoudre.

1. INTRODUCTION

Holophon est constitué de deux applications pour le contrôle de la spatialisation de multiples sources sonores vers un système de diffusion multi haut-parleurs.

« Holo-Edit » est un puissant éditeur graphique pour la programmation - la composition - de trajectoires sonores au moyen d'algorithmes génératifs ou transformatifs.

La spatialisation des trajectoires ainsi réalisées se fait au moyen d'« Holo-Spat », patch Max/MSP modulable optimisé pour la diffusion de multiples sources sonores permettant la spatialisation en temps réel.

Les deux applications communiquent au moyen de fichiers ou par communication inter-applications via MidiShare[1].

Le portage de ces applications sur Windows XP, dû à une demande croissante d'utilisateurs potentiels, au portage de Max/MSP sur cette même plate-forme ainsi qu'à la compatibilité grandissante entre les systèmes d'exploitation les plus courants (Apple, Microsoft et Unix) nous a également permis de mettre au point une version en ligne[2], multi-plateformes, reprenant la quasi-totalité des fonctionnalités de la version hors connexion.

Si le langage Java semble prévu à cet effet (la portabilité), nous allons voir que ce portage n'a pas été réalisé sans quelques difficultés.

2. GENERALITES SUR LE LANGAGE JAVA

2.1. Naissance et but du langage

Le langage Java naît dans les laboratoires de Sun au début des années 90 sur la base du c++ avec pour but la réalisation d'une technologie orientée multimédia en ligne et environnements réseau. Or, les systèmes d'exploitations de ces réseaux sont de plus en plus variés, créant des réseaux hybrides où les incompatibilités sont aussi diverses que nombreuses (gestion du système de fichier, formats et structures de données, gestions des périphériques…).

2.2. La Machine Virtuelle Java (JVM)

Pour pallier cette faiblesse apparente des réseaux, le langage n'a été conçu pour aucune plate-forme spécifique, mais plutôt comme une abstraction utilisant un code intermédiaire (appelé bytecode Java) traduit par la suite sur chaque plate-forme par une Machine Virtuelle Java distincte. Il s'agit donc à la fois d'un langage compilé et interprété [1][8].

Cette abstraction va jusqu'à codifier une taille unique pour chaque type de données (une application ou une applet Java donnera un même résultat à un calcul donné quelle que soit la plate-forme, ce qui n'est par exemple pas le cas sous Max).

Cette abstraction, outre la portabilité, permet d'écrire un code :

- Orienté objet,
- Clair (pas de notion de pointeurs, d'allocation de mémoire…)
- Léger (quelques Ko pour une classe compilée),
- Sûr (chaque JVM s'assure que le code qu'elle va interpréter n'accède pas au système de fichier de manière anormale…), …

2.3. Limites et contraintes

Une application ou une applet[3] Java présente donc l'avantage d'être multi-plateforme, à condition qu'une Machine Virtuelle Java soit disponible pour la plate-forme en question.

Or, pour la quasi-totalité des plateformes, la société Sun développe elle-même la JVM nécessaire à la bonne exécution de code Java, assurant ainsi une compatibilité complète entre celle-ci.

[1] © Grame
[2] www.gmem.fr/holo-online/ (sous OsX, utiliser Safari)
[3] Application récupérée à distance au travers d'un navigateur Internet

Ce n'est pas le cas sous Macintosh, et Microsoft tend à adopter la même politique. La raison semble évidente : une application multi-plateforme ne peut tenir compte des spécificités – essentiellement ergonomique - de chaque plateforme.

À noter également, la Machine Virtuelle Java de Mac OsX est lente (plus lente que celle d'Os9) et peu performante (surtout pour qui ne possède pas un G5). Cette lenteur peut s'expliquer par la volonté d'Apple d'intégrer les applications Java dans l'interface utilisateur particulièrement soignée du système (« Aqua »). Nous les avons contacté au sujet d'éventuelles améliorations de ses performances, mais n'avons pas obtenu de réponse à ce jour.

3. PORTAGE DE MAC OS X VERS WINDOWS

3.1. Principales différences entre les systèmes d'exploitation

Parmi ces différences, on peut noter : le système de fichier ; la gestion des caractères et polices ; les attributs des composants graphiques ainsi que la manière dont s'effectue le rafraîchissement de l'interface graphique.

La différence essentielle et la plus contraignante est l'obligation sous Macintosh de référencer, d'enregistrer l'application auprès du système, de manière à pouvoir quitter l'application depuis le Dock (ou tout simplement via la combinaison pomme-Q) tout en obtenant une notification de la part du système nous permettant de demander à l'utilisateur s'il veut sauvegarder ses modifications.

Cette différence nous oblige à compiler deux versions distinctes de l'application, car elle fait appel à des classes Macintosh natives, évidemment indisponibles sous Windows [4].

3.2. Minimisation de l'utilisation de l'API Java Macintosh

De manière à simplifier la maintenance et la mise à jour des deux versions, nous avons minimisé l'utilisation de l'API Java Macintosh (en dehors de la fonction d'enregistrement auprès du système décrite précédemment).

Pour les fonctions ne permettant pas d'uniformisation d'écriture, nous avons utilisé une variable booléenne s'initialisant au démarrage de l'application à partir d'informations fournies par le système [4]. Ce booléen a par la suite permis :

- l'utilisation de caractères différents pour la gestion du système de fichier[1] ou les caractères spéciaux,
- l'intégration de l'interface graphique de l'application dans le système d'exploitation (menu dans la barre des menus sous Macintosh, menu en haut de la fenêtre sous Windows),
- la gestion des différentes tailles des composants graphiques (composants graphiques « Aqua » plus volumineux sous Mac Os X).

3.3. Communication inter-application via MidiShare[2]

Le séquenceur Midi intégré à « Holo-Edit » utilise les outils de développement de MidiShare. Ce séquenceur permet de jouer les trajets sonores en temps réel. Il communique directement avec le pilote MidiShare (cf. figure 1). Celui-ci redirige par la suite la séquence midi vers un port Midi physique ou virtuel et nous offre diverses possibilités :

- Travailler avec plusieurs machines via des ports Midi physiques (une machine pour la gestion des trajectoires, l'autre pour la spatialisation sonore en temps réel) ;
- Travailler sur la même machine via un bus virtuel de communication Midi inter-application[3] ;
- Exploiter les trajectoires d'« Holo-Edit » dans un but tout autre (par exemple le contrôle d'automations d'un séquenceur physique ou logiciel – nécessite l'adaptation des données midi 14 bits vers 7 ou 10 bits avec éventuellement un changement du numéro de contrôleur) ;
- À plus long terme, transporter cette séquence Midi sur des réseaux plus complexes et plus étendus en utilisant les versions LAN et WAN du pilote MidiShare.

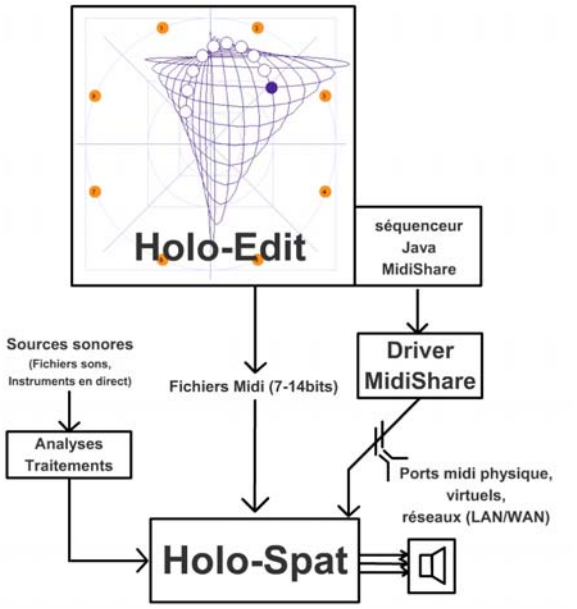

Figure 1. Communication entre Holo-Edit et Holo-Spat via Midishare

[1] Arborescence séparée par "/" sous Mac OsX et Unix, "\" sous Windows.

[2] © Grame

[3] IAC Bus sous Mac Os X, OMS sous Mac Os 9, Ports Midi virtuels de Max/MSP, Midi Yoke sous Windows XP…

MidiShare étant disponible pour tous les systèmes d'exploitation, le portage de cette communication n'a pas posé de problèmes particuliers.

Les performances sont par contre très différentes suivant la plateforme. Si Java est beaucoup plus performant sous Windows, les différents bus Midi inter-applications que nous avons pu tester semblent beaucoup moins performants (latence sensible, perte et confusion des données « inopinées ») que ceux offerts par Max/MSP sous Mac Os X.

4. PORTAGE D'APPLICATIONS VERS APPLET

4.1. But du portage

Les objectifs de ce portage pour le projet Holophon sont multiples :

- Rendre l'application « Holo-Edit » visible et disponible sur Internet, de manière à en augmenter la portée car elle constitue un outil performant et précis pour l'édition de trajets sonores.
- Donner la possibilité à l'utilisateur d'effectuer des corrections rapides d'une spatialisation préexistante sans nécessité d'installation.
- Quel que soit l'endroit, quelle que soit la machine, il suffit d'un navigateur[1] pour utiliser « Holo-Edit » et toutes ses fonctionnalités.
- À plus long terme, les possibilités sont diverses et variées, Internet ouvrant un champ nouveau à expérimenter. (cf. §5.1.4 Evolutions d'Holo-Edit-Online)

4.2. Java et la sécurité

4.2.1. Différences entre applets & applications

Les applets sont des applications Java intégrées dans des pages HTML, elles s'exécutent donc sur la machine cliente à partir du navigateur.

La structure d'une applet est légèrement différente de celle d'une application (gestion de l'initialisation, du rafraîchissement de l'interface graphique...) mais la différence fondamentale réside dans la gestion de la sécurité.

Dans l'absolu, un programme Java peut effectuer n'importe quelle opération sur la machine qui l'exécute, y compris l'effacement ou l'altération des données. S'il en était de même pour l'applet, elle pourrait corrompre l'intégrité du système à l'insu de l'utilisateur.

Afin d'éviter les actions malveillantes, même involontaires, les applets s'exécutent dans un environnement Java bridé. Les restrictions se situent au niveau des entrées/sorties, principalement :

- La lecture et l'écriture de fichiers sur la machine cliente
- La création de connexions (sockets) à des réseaux locaux ou distants.

[1] La plupart des navigateurs supportent le chargement d'Applet Java. Exception connue : Internet Explorer sous Mac Os9&X.

4.2.2. Complexité du système d'autorisation

Sans possibilité de sauvegarder ou d'importer des données, l'éditeur serait l'équivalent d'une version de démonstration de l'application.

La méthode la plus simple - pour le développeur - pour donner à l'Applet l'accès au système de fichier est l'utilisation de l'outil PolicyTool[3] inclut dans le Java Runtime Environment (JRE : environnement d'exécution). L'utilisateur doit lui-même créer une sorte de clé autorisant telle application à effectuer tel type d'opération (accéder à tel ou tel répertoire…).

Seulement, si la théorie semble simple, la pratique l'est beaucoup moins. Premièrement, cet outil n'est accessible qu'en ligne de commande à partir du terminal[2], ce qui repousse 90% des utilisateurs. Deuxièmement, il est d'une complexité telle (vocabulaire utilisé ; le nom des opérations à autoriser revient au nom des classes Java, c'est dire) que les 10% d'utilisateurs potentiels restant renoncent.

4.2.3. Solutions étudiées

Pour éviter à l'utilisateur ce type de manœuvre, nous avons étudié diverses propositions.

La première prévoyait la sauvegarde des données sur le serveur par l'application qui fournissait par la suite à l'utilisateur l'adresse à laquelle récupérer sa session de travail. L'applet n'ayant pas l'autorisation d'ouvrir de connexion réseau, elle ne peut donc pas communiquer avec le serveur par qui elle a transité.
La seconde affichait dans une fenêtre un texte contenant toutes les données de la session de travail de l'utilisateur. Il devait par ce biais, au moyen du « copier-coller » pour importer et exporter des données. Or, le même problème est apparu : l'applet ne peut accéder au presse-papier du système d'exploitation.

4.2.4. Solution adoptée : l'utilisation d'Applet signées

La solution adoptée est l'utilisation d'Applets signées. Le principe en est simple : l'auteur signe numériquement l'Applet. Par la suite, lors du son chargement sur la machine cliente, l'utilisateur est invité à valider un avertissement de sécurité (notifiant la provenance de l'application, les risques encourus… cf. figure 2).

Figure 2. Avertissement de sécurité lors du chargement de l'applet signée

[2] /usr/bin/policytool sous Mac Os X et c:\program files\Java\jre1.x.x\bin\policytool.exe sous Windows

Notons que s'il refuse cet avertissement, l'applet fonctionnera tout de même, mais il ne pourra ni sauvegarder, ni ouvrir de fichiers. Au contraire s'il accepte, toutes les fonctionnalités de l'applet seront disponibles, y compris les actions malveillantes, les restrictions décrites précédemment devenant caduques.

Mise en œuvre :

Une application Java est un ensemble de fichiers compilés (un par classe d'objet, ce qui permet la création de bibliothèque, une véritable programmation orientée objet). Pour la distribution des applications et des applets, ces fichiers sont compressés dans une archive JAR avec un fichier Manifest contenant le nom de la classe principale [8].

La signature numérique de l'Applet se fait au moyen des commandes keytool et jarsigner[3].

La commande keytool[1] permet la création d'une clé numérique dans laquelle l'auteur de l'application entre les informations décrivant l'application (version, numéros de série, durée de validité…) ainsi que son/ses auteurs (organisation, localisation géographique...).

Par la suite cette clé est insérée dans l'archive jar au moyen de l'outil jarsigner[2].

Par défaut la signature de l'archive supprime toutes les restrictions. Pour n'en autoriser que certaines, il faut utiliser l'outil policytool. (cf. §4.2.2 Complexité du système d'autorisation).

Notons que le message d'avertissement est généré en fonction de la clé numérique, il n'est pas programmé par le développeur (évitant ainsi des pratiques malveillantes).

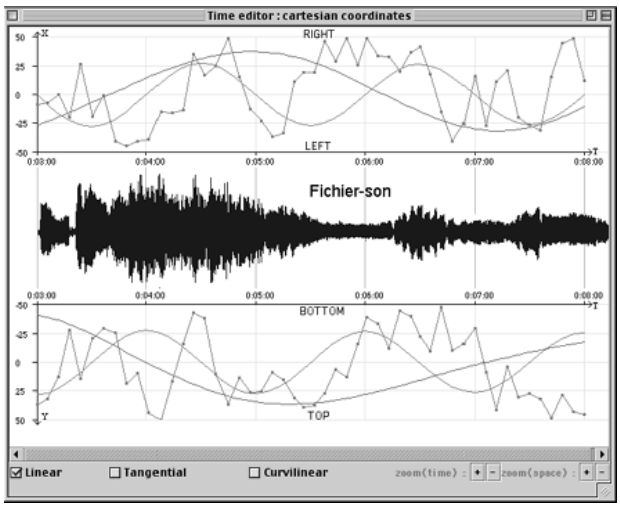

Figure 3. Intégration des formes d'ondes dans l'éditeur temporel de trajets sonores

[1] /usr/bin/keytool sous Mac Os X et c:\program files\Java\jre1.x.x\bin\keytool.exe sous Windows

[2] /usr/bin/jarsigner sous Mac Os X et c:\program files\Java\j2sdk1.x.x\bin\jarsigner.exe sous Windows

5. ÉVOLUTIONS : HOLOPHON 4

5.1. Holo-Edit

Les éditeurs temporels, permettant la création d'accélération, de ralenti, donnant une dynamique essentielle au trajet sonore, ne sont pas très fonctionnels actuellement, ils méritent une ré-écriture.

Par la suite, nous prévoyons l'intégration dans ces éditeurs d'un affichage de la forme de l'onde sonore qui est destinée à être spatialisée, de façon à suivre son évolution (cf. figure 3).

D'autre part, les trajectoires sont actuellement définies sous forme d'un vecteur (liste de points) pour chaque piste. Or, par sa nature, une piste audio est généralement formée d'une succession de sons. Nous envisageons donc de construire un éditeur permettant de manipuler des segments de trajectoires (déplacement graphique à la souris, duplication etc.). Il s'agit alors d'ajouter une fenêtre de montage à l'application, permettant la visualisation et la manipulation d'objets sonores (forme d'onde associée au mouvement, cf. figure 4).

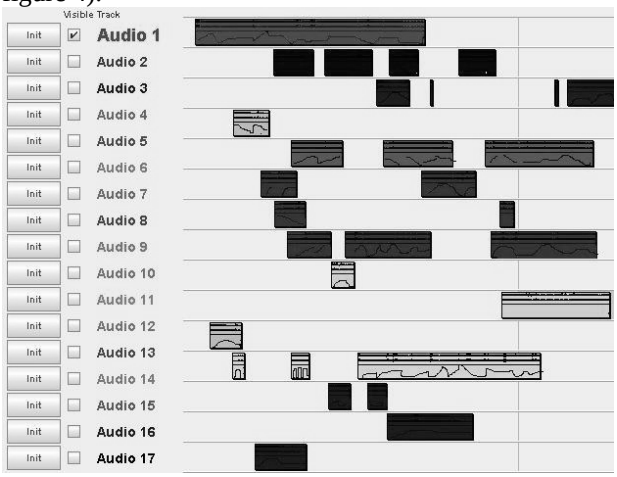

Figure 4. Fenêtre de montage prévisionnelle de l'éditeur. Formes d'onde et trajets sont associés pour former des objets sonores

5.2. Holo-Spat

L'évolution d'Holo-Spat prévoit la conception d'algorithmes haut-niveau pour le contrôle en temps-réel de la spatialisation. Cette évolution sera particulièrement adaptée à la spatialisation d'instruments en direct.

La configuration standard du système de diffusion prévue par Holophon est relativement large (disposition circulaire ou latérale de 4 à 16 enceintes, cube ou 5.1). Il n'est toutefois pas rare de devoir apporter des corrections en fonction de la salle de concert et du système de diffusion. C'est pourquoi la prise en compte de la position des haut-parleurs (position actuellement éditable dans Holo-Edit) est également prévue.

De manière à intégrer plus aisément la spatialisation au processus de composition et pour répondre à l'attente de nombreux compositeurs, nous allons réaliser une version plugin (VST) d'Holo-Spat, compatible avec

Holo-Edit. Ce plugin permettra l'utilisation directe de nos outils dans des logiciels de montage audionumérique standards (Performer, Cubase, Logic…).

5.3. Communication inter-application

D'autres moyens de communication entre Holo-Edit et Holo-Spat sont à l'étude pour les prochaines versions d'Holophon :

- Utilisation d'OpenSoundControl[1] (OSC, cf. figure 5.a),
- Intégration de l'éditeur dans Max/MSP via mxj & mxj~, Kit de développement d'objets externes en Java (cf. figure 5.b).

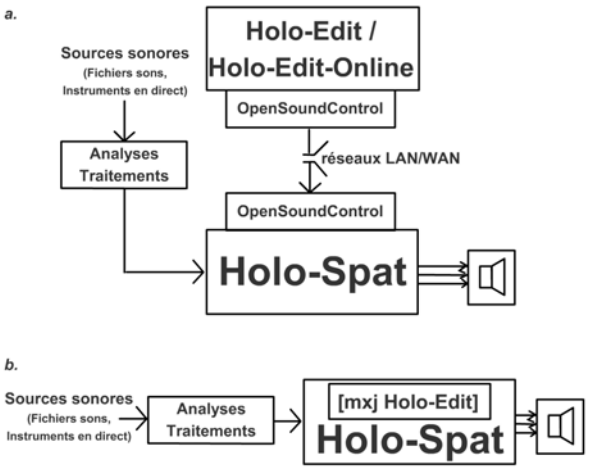

Figure 5. (a) Communication via OSC (b) Editeur intégré au patch Max/MSP via l'objet

5.4. Holo-Edit-online

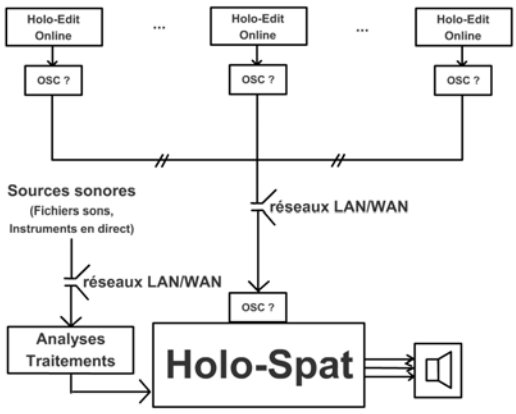

Figure 6. Utilisation possible de la future version en ligne d'HoloEdit

Une des évolutions évidentes d'Holo-Edit-online est l'exploitation des possibles d'Internet en matière de contrôle à distance d'une spatialisation. On pourrait par exemple imaginer des installations spatialisées contrôlées par plusieurs utilisateurs, chacun contrôlant une partie différente de l'installation ou la réalisation d'un serveur de trajectoires sonores… (cf. §3.3 Communication inter-application et figure 6).

[1] Protocole de transport de contrôles pour interactions entre des applications multimédias, capteurs et chaînes de traitement du signal. Idéal pour les réseaux hybrides, développé dans la quasi-totalité des langages, notamment en Java et Max/MSP. Open-source, initié par le CNMAT, Berkeley, University of California.

6. CONCLUSION

Holophon a déjà permis à de nombreux artistes d'expérimenter l'espace sonore, et d'intégrer la spatialisation sonore à l'écriture musicale. Ce procédé d'intégration nécessite l'évolution et l'amélioration constante des outils existants par leur adaptation aux besoins spécifiques d'une œuvre mais également par l'ajout de fonctionnalités ouvrant de nouveaux champs, offrant une liberté toujours plus grande à l'artiste. L'utilisation des réseaux et des communications à distance, partie essentielle de ce processus, introduit des possibilités nouvelles qu'il nous faut étudier.

7. REFERENCES

[1] Ken Arnold, James Gosling et David Holmes, « Le langage Java », Vuibert, Paris, 2001

[2] Benjamin Cabaud, Laurent Pottier, « Le contrôle de la spatialisation multi-sources. Nouvelles fonctionnalités dans Holophon version 2.2 », *Actes des Journées d'Informatique Musicale (JIM02)*, Gmem, Marseille, 2002.

[3] Mary Dageforde, « Security in Java 2 SDK 1.2 », *The Source for developpers – Security Tutorials*, Sun MicroSystem Inc., http://Java.sun.com/docs/books/tutorial/security1.2/index.html

[4] Léopold Frey, « Portage d'Holophon, ensemble d'outils pour la spatialisation », *Projet de fin d'étude*, Utbm, Belfort, 2004.

[5] Laurent Pottier, « Dynamical spatialization of sound. HOLOPHON : a graphic and algorithmic editor for Sigma1 », *Dafx98 Proceedings*, Barcelone, 1998.

[6] Sun MicroSystem Inc., « Summary of Tools for the JavaTM 2 Platform Security », *The Source for developpers – Security Tools Summary*, Sun MicroSystem Inc., http://Java.sun.com/j2se/1.4.2/docs/guide/security/SecurityToolsSummary.html

[7] Sun MicroSystem Inc., « JAR File Specification », *The Source for developpers – Security Tools Summary*, Sun MicroSystem Inc., http://Java.sun.com/j2se/1.3/docs/guide/jar/jar.html

[8] Auteurs multiples, « Java (langage) », *Wikipedia – L'encyclopédie libre*, http://fr.wikipedia.org/wiki/Java_(langage)