

# MMS

## Un séquenceur multimédia dans Max-MSP

Mémoire de CECS en  
« Recherche appliquée en électroacoustique  
et informatiques musicales »

Conservatoire National Supérieur  
de Musique et Danse de Lyon

-----  
Département SONVS  
Année 2006-2007

TWOROWSKI Sébastien



=====

== Plan Du Mémoire ==

=====

*** Avant propos	7
Conventions concernant les polices de caractères	8
I- Comment est venu le projet, comment est-il né ?	9
II- Le choix de l'environnement de programmation Max-MSP	11
II.1- Un rapide aperçu de Max-MSP	11
II.2- Max-MSP pour développer le projet	11
III- Une brève histoire de Max-MSP	13
IV- Le choix de développer un logiciel libre	14
IV.1- Que signifie l'expression « logiciel libre »	14
IV.2- Les racines du logiciel libre	15
IV.3- Quelques qualités des logiciels libres	16
IV.4- Max-MSP, un logiciel libre ?	16
IV.5- « Tout seul on va plus vite mais ensemble on va plus loin »	17
V- Journal de recherches	18
V.1- Au commencement était MOELLER...	18
V.2- Définir un type d'interface utilisateur	19
V.3- Le modèle du séquenceur	20
V.4- Le son	23
V.5- Une interface graphique pour les fichiers audio	24
V.6- Alzheimer chez les objets graphiques	25
V.7- Une nouvelle version	27
V.7.1- La fenêtre « <i>inspector</i> »	28
V.7.2- Le pointeur de lecture	30
V.7.3- Patcher son et pointeur de lecture	30
V.7.4- La mémoire -> <b>coll</b>	31
V.7.5- Les modes	33
V.7.6- Bilan de cette version	35
V.7.7- L'objet <b>timeline</b>	36
V.7.8- L'apocalypse selon <b>timeline</b>	37
V.8- Vers une version évolutive pour <i>plug-ins</i>	37
V.9- Bilan de cette version 060325 et orientation vers une version finale	41
VI- Renseignements techniques concernant les périphériques externes	43
VI.1- La carte GLUION	43
VI.2- Explications techniques et électroniques pour l'utilisation de la carte GLUION	44
VI.3- Quels sont les impératifs d'un relais mécaniques ?	47
VI.4- Qu'est-ce qu'un transistor ?	49
VI.5- Protocoles de communication	51
VI.5.1- Définition du mot protocole selon le petit Robert	51
VI.5.2- Définition d'un protocole de communication selon Wikipédia, l'encyclopédie libre sur internet	51
VI.5.3- Le concept	52
VI.5.4- Qu'est-ce que Ethernet ?	52
VI.5.5- Revenons au patch	53
VI.5.6- Qu'est-ce qu'UDP ?	54
VI.5.7- Qu'est-ce que le DMX ?	54
VII- Version actuelle	56
VII.1- Installation de notre séquenceur multimédia	56
VII.2- Configurer Max-MSP pour l'utilisation de MMS	58

VII.3- Comment paramètriser votre système d'exploitation pour l'utilisation de MMS	59
VII.3.1- Préférences système sous Mac OS X	59
VII.3.2- Configuration réseau pour Windows XP	60
VII.4- Comment l'application MMS fonctionne-t-elle en interne ?	61
VII.4.1- Pourquoi un <i>template</i> ?	61
VII.4.2- Fermer le mode édition d'un <i>template</i> Max-MSP	62
VII.4.3- Description de la fenêtre principale MMS	64
VII.4.3.1- Une seule et même mémoire par session	65
VII.4.4- Description de la fenêtre de création de <i>plug-ins</i>	68
VII.4.5- Description de la fenêtre <i>Waveform</i> (forme d'onde)	73
VII.4.5.1- Édition des données propres à un <i>plug-in</i> particulier	76
VII.4.5.2- Sélection d'un <i>plug-in</i> pour édition	77
VII.4.6- Explication de la fenêtre Destruction	78
VII.4.7- Description de la fenêtre de zoom ( <i>Zoom-Time</i> )	79
VII.4.8- Description de la fenêtre de montage ( <i>Events</i> )	80
VII.4.9- Description des différents <i>plug-ins</i>	84
VII.4.9.1- Le <i>plug-in</i> de type contact	84
VII.4.9.2- Description du <i>plug-in</i> pour le contrôle des sorties audio ( <i>SoundLoudSpeaker</i> )	87
VII.4.9.3- Description du <i>plug-in</i> vidéo	89
VII.5- Lecture du montage	90
VII.5.1- Jeux_CONTACT	91
VII.5.2- Jeux_GRADATEUR	92
VII.5.3- Jeux_VIDEO	94
VII.5.4- Jeux_SONHP	95
<b>VIII- Création d'une nouvelle session MMS pas à pas</b>	96
VIII.1- Créer une nouvelles session de travail	96
VIII.2- Charger un fichier audio dans la session	96
VIII.3- Créer un <i>plug-in</i> dans la fenêtre de montage	97
VIII.4- Éditer le contenu d'un <i>plug-in</i>	97
VIII.5- Sauvegarder la session pour la première fois	98
VIII.6- Ouvrir une session existante	98
<b>IX- Perspectives d'évolutions pour l'application MMS</b>	99
IX.1- La vidéo	99
IX.2- Le son	99
IX.3- L'interface utilisateur	100
IX.4- Au-delà du CNSMD	100
<b>Références</b>	101
<b>Remerciements</b>	102
<b>Annexes</b>	103
Quelques mots sur mon expérience d'assistant musical au FNM de Stuttgart	105
<b>Photos</b>	107
- ScriptCOMandRec	109
- Lcdeplace	110
- multiCanal	111
- interf_proviNew	112
- demo060325	113
- OSC_Basics	114
- regle_interne	115





## Avant-propos :

Le logiciel présenté dans ce mémoire fut développé dans un premier temps pour des besoins tout à fait particuliers et propres à un projet donné : le projet nommé « Vignéroscope » dont Monsieur BERARD est le concepteur :

[http://www.lesvinsfrancais.com/editoriel.php?code\\_dossier=76&lsc=74x76&c\\_t=1&c\\_c=147&lang=](http://www.lesvinsfrancais.com/editoriel.php?code_dossier=76&lsc=74x76&c_t=1&c_c=147&lang=).

Cependant, dès le début, ma véritable intention fut de saisir cette occasion pour créer un séquenceur pour la création multimédia, la création musicale et le théâtre musical. C'est-à-dire permettre l'agencement, le déclenchement et la synchronisation d'événements musicaux avec tous autres types d'événements multimédia. Par ex. :

- musique, danse et lumières pour un spectacle chorégraphique
- spectacles et installations créés par des artistes multimédia
- environnements audio et vidéo appliqués à la pédagogie
- concert de musique mixte dans le cas où le compositeur voudrait également inclure la gestion des lumières et de la vidéo à sa création artistique
- ...etc.

Le contenu de ce mémoire implique quelques points importants concernant le lecteur :

- vous devez être à l'aise avec l'utilisation d'un ordinateur sous les systèmes d'exploitation Mac OS X ou Windows XP
- être familiarisé avec l'environnement de programmation Max-MSP. Il n'est nullement nécessaire que vous en ayez la parfaite maîtrise mais il est recommandé que vous en connaissiez les principes de base
- afin de pouvoir tester les différents patchers qui correspondent aux multiples exemples joint à ce travail, vous devez avoir une version de l'environnement de programmation Max-MSP supérieure ou égale à 4.6 installée sur votre ordinateur. Si ce n'est pas le cas, sachez néanmoins qu'il vous est possible d'en télécharger une version de démonstration valable trente jours à partir de l'URL suivante :

<http://www.cycling74.com/downloads/maxmsp>

Cette version de démonstration donne accès à toutes les fonctions de l'environnement et n'est aucunement limitée

Vous n'êtes pas tenu d'utiliser les patches fournis en exemple pour suivre le déroulement du mémoire. Je les ai inclus afin qu'ils puissent éventuellement être utiles au lecteur et que tous les « fichiers source » soient accessibles.

Vous pouvez bien entendu réutiliser chacun des différents patches selon les directives spécifiées dans la licence livrée avec ce programme (voir CD-Rom).

Conventions concernant les polices de caractères :

- les mots anglais sont en *Arial italique*
- les noms d'objets sont en **Courier gras**
- les messages Max et scripts Max sont en Courier new
- les citations sont simplement en italiques entre guillemets
- Les noms de dossiers et fichiers sont en cochin

Pour toute question, suggestion ou encore réflexion relative à ce mémoire, n'hésitez pas à me contacter par courrier électronique à : [sebastien.tworowski@laposte.net](mailto:sebastien.tworowski@laposte.net)



## I- Comment est venu le projet, comment est-il né ?

C'est en octobre 2005, peu de temps après la reprise des cours au Conservatoire National Supérieur, que Monsieur Denis LORRAIN nous a fait part du désir de Monsieur BERARD d'entrer en contact avec un étudiant capable de développer un logiciel informatique pour un projet en Californie. Le projet, à cette période n'était que très flou et il convenait à la personne intéressée de prendre contact directement avec Monsieur BERARD afin de définir les besoins et les délais. Tout ce que je savais du projet à l'époque, c'est qu'il devrait consister en la diffusion de son sur plusieurs points. J'ai donc pris rapidement contact avec Monsieur BERARD mais je n'imaginais pas à cet instant ce qui allait découler de cette rencontre.

Début novembre, Monsieur BERARD m'en disait plus. Il était à la recherche d'un développeur informatique ayant de bonnes connaissances en audio, vidéo, MIDI, montage son et vidéo ainsi que d'une personne qui pourrait le conseiller sur le matériel à utiliser en général.

Monsieur BERARD est à l'origine d'une idée de mise en scène d'une importante collection d'objets particuliers au monde viticole. Ces objets sont exposés au public dans un musée et mis en scène au travers de jeux de lumières, de vidéo projections et bandes audio racontant leur histoire, leur mode d'utilisation et quelques anecdotes. Monsieur BERARD n'est plus exploitant du musée ni du vignoble auquel cette exposition était rattachée. Il a vendu le concept à un exploitant en Californie qui désire reprendre le spectacle et l'intégrer à son propre musée. Le matériel électronique et tout le système informatique nécessitaient cependant une complète remise à jour. C'était par là aussi l'occasion de tout remettre en question et d'envisager quelque chose d'entièrement neuf tant au niveau du scénario que de tout ce qui pilotera le déroulement du spectacle.

L'idée à ce moment là était de pouvoir profiter des avancées technologiques en matière de contrôle du son, de la lumière et de la vidéo au sein d'un ancien programme très rudimentaire fait à base de quelques lignes de commandes en MS-DOS (*Microsoft Disk Operating System*). Ce programme, bien trop rudimentaire ne permettait pas la réalisation de tâches multiples en simultané. Pour lire un fichier audio, il fallait par exemple que l'ordinateur interrompe toute autre tâche en cours. Quand à la vidéo... ce n'était qu'un rêve.

Le projet s'annonçait donc plus ambitieux qu'à l'origine. Cela ne me surprit pas cependant. Bien au contraire, j'ai tout de suite vu en ce projet quelque chose de bien plus large et flexible qu'une utilisation uniquement limitée à celle de ce musée. C'était aussi pour moi une opportunité de plus d'être en contact avec le monde extra-scolaire sur un projet concret de développement informatique multimédia.

J'ai donc accepté de travailler sur ce projet mais avant de pouvoir aller plus loin, un point d'importance majeure restait à éclaircir ; à savoir : sera-t-il possible de communiquer avec les contacts MOELLER via Ethernet en utilisant l'environnement de programmation Max-MSP ou devrai-je recourir à un langage de programmation particulier comme C ou javascript ?

Dans l'ancienne version du programme, la version MS-DOS, le déclenchement des projecteurs se faisait avec des contacts MOELLER. Cette version des contacts MOELLER est bien entendu aujourd'hui obsolète. D'autres contacts sont aujourd'hui disponibles et sont contrôlables par Ethernet. Cependant, d'autres matériels existent, ils sont à la fois plus flexibles et moins coûteux.

## II- Le choix de l'environnement de programmation Max-MSP

### II.1- Un rapide aperçu de Max-MSP

Max-MSP est un environnement de programmation graphique pour la musique, l'audio et le multimédia. Cet environnement est utilisé de par le monde depuis plus de quinze ans par des musiciens, compositeurs, artistes, professeurs et étudiants pour réaliser des projets qui reflètent leurs idées personnelles. Max-MSP est un vrai système de développement compatible Mac OS X et Windows XP. Avec Max-MSP, il est possible de créer nos propres logiciels applicatifs en utilisant un kit graphique d'objets et en les connectant entre eux à l'aide de connexions (dicordes) appelées *patch cords*.

Max-MSP se divise en fait en deux parties distinctes :

- **Max** est l'environnement de base. C'est avec Max que nous pouvons piloter de façon très large des instruments MIDI ou générer des messages de contrôle... Nous avons également toute une palette d'objets dits "interface utilisateur" comme des courbes réglables qui peuvent servir, par exemple, à contrôler la largeur de bande d'un filtre audio... etc.
- **MSP** est la "boîte à outils" qui permet de faire du traitement audio. Il est possible de tout faire en matière de traitement audio avec MSP, du filtre interactif à l'enregistrement *direct to disk*, par exemple, et bien d'autres choses encore.

A Max et MSP peut encore s'ajouter une extension supplémentaire appelée Jitter :

- **Jitter** est aussi un environnement graphique, totalement intégré comme troisième couche de Max qui permet, entre autres, le traitement en temps réel de la vidéo et du graphisme en 3D (trois dimensions).

### II.2- Max-MSP pour développer le projet

Même à partir de cette très rapide et sommaire description, il semble tout de suite évident que Max-MSP et éventuellement Jitter forment l'environnement de programmation idéal pour la réalisation d'un tel projet. Nous aurons également la possibilité de le porter au delà des besoins de Monsieur BERARD, et de rendre l'application résultante plus flexible et généralisable, utilisable dans de nombreux domaines, dont la musique pure.

Max-MSP est un environnement fiable, très puissant et sans réelle limite. Comme il est dit plus haut, Max-MSP est constitué d'objets que l'on connecte entre eux afin de construire une application finale. Il existe un choix important d'objets fournis avec le logiciel. Ces objets permettent de réaliser déjà pratiquement tous les projets imaginables s'ils sont utilisés de manière intelligente, cependant, pour des raisons personnelles ou un besoin bien particulier, il est tout à fait possible (moyennant quelques connaissances en C ou javascript par exemple) de créer ses propres objets.

Une importante communauté de développeurs s'est créée autour de Max-MSP et Jitter. Ces échanges permettent d'obtenir des conseils auprès d'autres programmeurs mais aussi les dernières informations, la connaissance de *bugs*... etc. Ceci est très important.

Max-MSP permet de modéliser et tester rapidement une idée et de décider de sa viabilité. Cette méthode de programmation permet une interaction entre le logiciel et ses résultats (même grossiers) qui permet de voir rapidement quelle direction prendre. Il n'y a pas besoin de compilation du code comme en C par exemple. Le patch(1) "tourne" (fonctionne), fait ce que les objets qui y sont présents lui permettent de faire alors qu'en même temps, il est tout à fait possible d'effectuer des changements à l'intérieur du patch lui-même. Les développeurs mais aussi les compositeurs sont toujours fascinés de voir le résultat changer sous leurs yeux de manière aussi rapide et efficace en temps-réel. Ceci est très intéressant pour un développeur, ces possibilités donnent à Max-MSP une force extraordinaire que beaucoup d'autres environnements ou langages de programmation n'ont pas.

Grâce à cette profusion d'objets mais aussi à l'immense flexibilité qu'offre la possibilité d'en programmer soi-même, Max-MSP peut recevoir ou envoyer des messages de/vers n'importe quel périphérique externe. Il suffit au développeur de savoir comment convertir le signal entrant en messages compréhensibles par Max ou inversement de savoir comment construire à l'aide de Max des messages à destination des périphériques souhaités. Nous pouvons communiquer par Ethernet, MIDI, *blue tooth*, DMX(2)... etc.

Bien entendu, un tel environnement n'est pas né en un jour, il aura fallu des années de développement et de tests pour en arriver à l'état actuel des choses. État qui ne cesse d'évoluer de manière positive. Afin de mieux comprendre comment Max-MSP en est arrivé là, je vous propose de lire un résumé de l'histoire de Max. (Le lecteur ayant déjà connaissance de l'histoire de Max pourra bien entendu sauter cette section.)

1- Le mot patch définit la globalité de l'application développée en Max-MSP/Jitter. Vous trouverez également le mot patcher qui, lui, ne définit qu'une fenêtre unique de l'application, une partie du programme principal.

2- Ce sigle sera explicité plus loin dans le chapitre VI.5.7

### III- Une brève histoire de Max-MSP

(Traduit de l'anglais à partir de l'URL suivante :

[http://www.cycling74.com/twiki/bin/view/FAQs/MaxMSPHistory#Where\\_did\\_MaxMSP\\_come\\_from.\)](http://www.cycling74.com/twiki/bin/view/FAQs/MaxMSPHistory#Where_did_MaxMSP_come_from.)

Max fut d'abord développé pour Macintosh dans le milieu des années 1980 par Miller Puckette. Son origine première serait un logiciel nommé *Patcher*, également conçu par Miller Puckette. Ce logiciel *Patcher* permettait de traiter et de contrôler des événements MIDI. En 1989, l'IRCAM commençait un projet basé sur une carte fabriquée par Ariel permettant de faire de la synthèse en temps-réel sur les ordinateurs NeXT. Ce projet s'appelait alors *IRCAM Signal Processing Workstation* (ISPW). Miller Puckette porta alors Max pour la station NeXT et ISPW. Il y ajouta une batterie d'objets pour faire du traitement audio sur cette carte. Max combiné avec l'audio était connu sous le nom de Max/FTS et était largement utilisé à l'IRCAM et dans une trentaine d'autres centres, dont SONVS au CNSMD de Lyon, ainsi que dans des studios individuels à travers le monde. A cette époque, une seule carte avec deux processeurs coûtait environs 12.000 dollars sans même compter l'ordinateur NeXT devant l'accueillir.

Max, en version Macintosh, fut originellement distribué dans le commerce par *Opcode Systems, Inc.* en 1990. Depuis 1999, Max est distribué et développé par Cycling '74.

En 1996, Miller Puckette, alors à l'université de Californie à San Diego, commençait à développer un nouveau programme (*Pd* ou *Pure Data*) destiné à une nouvelle génération de microprocesseurs plus puissants. Les buts de *Pd* sont différents de ceux de Max/FTS mais les deux logiciels ont en commun d'offrir la possibilité de faire du traitement du signal en temps réel en connectant des objets les uns aux autres. Initialement développé sur les stations SGI, *Pd* fonctionne maintenant sous Windows, Linux et Mac OS X. Peu de temps après le lancement de *Pd*, David Zicarelli décidait d'ajouter le traitement audio pour les ordinateurs de type Macintosh Power PC à la version existante de Max distribuée par Opcode. Max-MSP en est le résultat.

Max-MSP utilise l'infrastructure de traitement du signal de *Pd* et y ajoute des caractéristiques inspirées de Max/FTS mais pas encore présentes dans *Pd*. A cela s'ajoutent d'autres innovations ainsi que des extensions "interfaces utilisateur" appropriées à l'environnement Max-MSP.

Pour d'autres informations sur l'environnement Max, je vous conseille le site de Cycling '74 :

<http://www.cycling74.com>

## IV- Le choix de développer un logiciel libre

Le logiciel décrit dans le présent mémoire de CECS au CNSMD de Lyon est un logiciel dit "logiciel libre".

### IV.1- Que signifie l'expression "logiciel libre" ?

"Libre" correspond à une licence de logiciel obéissant à une définition très précise établie par la *Free Software Foundation* (FSF), dont voici les principaux critères nécessaires :

- la libre redistribution
- un code source disponible
- les travaux dérivés possibles

Le fait de disposer des sources d'un logiciel ne suffit pas à dire qu'il est "libre". Dans tous les cas, on se référera à la licence d'utilisation du logiciel. Pour une définition détaillée du logiciel libre voir l'hyperlien suivant :

[http://fr.wikipedia.org/wiki/Logiciel\\_libre#Qualit.C3.A9s\\_des\\_logiciels\\_libres](http://fr.wikipedia.org/wiki/Logiciel_libre#Qualit.C3.A9s_des_logiciels_libres). Mais aussi, voir la licence GPL délivrée avec le logiciel (lecture plus que recommandée).

Un logiciel libre est un logiciel dont la licence permet à tout acquéreur de l'utiliser, l'étudier et le modifier sans restriction. Chacun est aussi libre de le redistribuer sans restriction, en respectant des contraintes éventuelles visant à garantir la perpétuation de son caractère libre.

La notion de logiciel libre a été formalisée dans la première moitié des années 1980 par Richard Stallman. Il l'a popularisée avec le projet GNU(1) et la *Free Software Foundation* (FSF). Le logiciel libre le plus connu est Linux. Depuis la fin des années 1990, le succès des logiciels libres suscite un vif intérêt dans l'industrie informatique et les médias. Il ne faut pas confondre les logiciels libres avec les logiciels gratuits (*freewares*), ni avec les *sharewares*, ni avec des logiciels tombés dans le domaine public. La notion de logiciel *Open Source* établie par l'*Open Source Initiative* est en revanche très proche de celle de logiciel libre.

1- GNU est un acronyme récursif pour « GNU's Not UNIX » (littéralement : GNU N'est pas UNIX). Au début de la création de GNU, le système d'exploitation UNIX était déjà largement répandu, et il était généralement admis par les informaticiens que son architecture avait fait ses preuves. GNU fut donc conçu pour être compatible avec ce système. On ne peut comprendre réellement ce qu'est le projet GNU si l'on néglige ses motivations, relevant de l'éthique et de la philosophie politique. Il vise en effet à ne pas laisser l'homme devenir l'esclave de la machine et de ceux qui auraient l'exclusivité de sa programmation. Le projet GNU œuvre pour une libre diffusion des connaissances, ce qui n'est pas sans avoir d'importantes implications politiques, éthiques et philosophiques.

La licence libre la plus connue est la Licence Publique Générale GNU, mais il existe aussi d'autres licences, spécifiquement créées pour certains domaines très divers (art, jeux de rôle, revues scientifiques, etc.), qui peuvent être considérées comme des « licences *CopyLeft* ».

L'expression *CopyLeft* est un jeu de mot par opposition à copyright (*right* = droit (avoir un droit...) ou droite (opposé à gauche)). *CopyLeft* signifie donc littéralement « copie gauche ». C'est la possibilité donnée par l'auteur d'un travail soumis au droit d'auteur (œuvre d'art, texte, programme informatique, etc.) à l'utilisateur de copier, utiliser, étudier, modifier et distribuer son œuvre, avec la restriction que celui-ci devra en perpétuer les mêmes conditions d'utilisation, y compris dans les versions modifiées ou étendues. Autrement dit, l'utilisation du *CopyLeft* est "contagieuse". Toutes les licences de logiciel libre ne sont pas basées sur le principe du *CopyLeft*. Il n'y a alors pas de contagion obligatoire, et le logiciel libre peut être exploité par des entreprises privées.

Le logiciel que je me propose de décrire ici est quant à lui distribué sous Licence Publique Générale *CopyLeft*.

#### **IV.2- Les racines du logiciel libre**

Le partage de logiciel existe pratiquement depuis aussi longtemps que les logiciels eux-mêmes. Dans les débuts de l'informatique, les fabricants pensaient que la concurrence était située essentiellement sur le plan du *hardware* (la machine). De ce fait, ils ne portaient pas une grande attention au *software* (logiciel) en tant que source de revenus.

La plupart des acheteurs de ces premières machines étaient essentiellement des scientifiques ou des techniciens qui étaient tout à fait capables de modifier et accroître les possibilités du logiciel livré avec la machine selon leurs besoins particuliers. Ces mêmes acheteurs redistribuaient parfois leurs modifications non seulement au constructeur mais également à d'autres possesseurs de machines similaires. Les fabricants quant à eux toléraient et même encourageaient ces pratiques : à leurs yeux, les améliorations apportées à la partie logicielle, d'où qu'elles viennent, ne faisaient que rendre la machine plus attractive aux yeux d'éventuels nouveaux acheteurs.

Ce mode de gestion du logiciel est resté viable tant que chaque fabricant de matériel n'utilisait que son propre système d'exploitation (et langage « assembleur » de bas niveau), incompatible avec celui des concurrents. Il ne s'appliquait cependant pas à des développements liés à des langages évolués et répandus (Fortran, par exemple) qui étaient, eux, relativement universels.

L'apparition de logiciels libres, par opposition à des logiciels dits « propriétaires » est due à une standardisation centrée sur un petit nombre de systèmes d'exploitation (Windows, Linux, Mac OS...) en général compatibles avec les matériels de nombreux fabricants.

#### IV.3- Quelques qualités des logiciels libres

Ces logiciels libres présentent les principaux avantages suivants :

- le développement coopératif entre plusieurs programmeurs entraîne une bonne rédaction et documentation du code informatique, afin de faciliter les lectures et contributions extérieures
- la qualité est souvent proportionnelle au nombre des développeurs. Plus la communauté de développement s'étend, plus elle devient un gage de qualité et de réactivité. De la même manière, la communauté des utilisateurs, ayant comme rôle principal de faire remonter des dysfonctionnements et des suggestions, a une influence proportionnelle à sa taille
- les logiciels libres peuvent offrir des garanties de sécurité supérieures à celles des logiciels propriétaires :
  - examen préalable du code source du logiciel par des experts
  - impossibilité d'avoir recours à la sécurité par l'obscurité : en disposant des sources, il est souvent plus rapide de maintenir le niveau de sécurité (implémentation de nouvelles mesures, correction d'une éventuelle faille, ...)

#### IV.4- Max-MSP, un logiciel libre ?

Max-MSP n'est pas vraiment un logiciel libre... Je le qualifierais de logiciel semi-libre. Non libre car pour pouvoir utiliser Max-MSP ou même seulement Max, il faut acheter une licence d'utilisation. Certaines mises à jour sont aussi payantes et leurs coûts peuvent même être relativement élevés. Enfin, il n'est absolument pas possible de redistribuer Max-MSP.

Cela dit, d'une certaine façon, Max-MSP et même Jitter ont une franche ouverture au public. Cette ouverture donne une très grande puissance et flexibilité à cet environnement de programmation. Il est tout à fait possible pour chaque utilisateur Max ayant une connaissance d'un langage de programmation tel le C, de programmer ses propres objets dont il pourra à son tour, si le coeur lui en dit, faire profiter toute la communauté. Il est possible de télécharger un kit de développement à partir du site internet de Cycling '74. Ce kit de développement est extrêmement bien documenté. Cette documentation est accompagnée de nombreux codes source d'objets natifs aussi bien pour Max que pour MSP (et même Jitter dans un autre kit de développement (*Software Development Kit* (SDK))). Chaque code source est documenté afin de servir d'exemple. Il est tout à fait autorisé de copier ces codes source afin de les réutiliser dans d'autres projets ou dans le but de les améliorer ou encore adapter un objet en fonction de besoins bien précis et tout à fait particuliers à un projet.

Il existe des *mailing lists* consacrées aux discussions et questions d'utilisateurs mais aussi des listes destinées aux développeurs désireux de programmer leurs propres objets Max, MSP ou encore Jitter. Ces *mailing lists* sont des lieux d'échanges importants tant au niveau des discussions entre utilisateurs qu'en matière de codes source. La majorité des développeurs Max-MSP mettent volontiers leurs travaux à la disposition de toute



personne et ceci dans le but et avec le simple espoir qu'ils puissent être utiles à d'autres.

Chaque programme développé à partir de l'environnement Max-MSP peut cependant devenir une application de type propriétaire c'est à dire, vendue sous licence utilisateur sans avoir à verser de droits à Cycling '74, ni à Opcode et ni à l'IRCAM. C'est au développeur d'en faire le choix, il est décideur du devenir de sa création.

#### **IV.5- "Tout seul on va plus vite mais ensemble on va plus loin"**

Le logiciel présenté dans ce travail de recherche et de programmation informatique a été développé au Conservatoire National Supérieur de Musique et Danse de Lyon. Il peut-être considéré comme un travail universitaire réalisé lors de mes études dans cet établissement. En tant que travail universitaire fait par un étudiant dans le cadre de ses études, le fait que le logiciel puisse être étudié, modifié et que son code source soit disponible à tous dans un but pédagogique, me semble parfaitement justifié.

Il est évident que sans l'aide et l'appui de François ROUX (professeur Max-MSP au CNSMD de Lyon), le développement de ce logiciel ne serait certainement pas allé aussi loin.

Ce développement informatique multimédia ne doit pas s'arrêter à mon départ du CNSMD de Lyon. Quiconque voudra étudier ce programme et y apporter des améliorations est bien entendu libre de le faire. Il me semble important que si Monsieur ROUX désire utiliser les codes source de ce logiciel à des fins personnelles ou pédagogiques, qu'il puisse le faire. De même, si un compositeur voulait utiliser le programme pour une réalisation, il est important qu'il puisse le faire librement et qu'il puisse également y ajouter des fonctions supplémentaires si celles déjà présentes venaient à ne pas être suffisantes (ex. : plus de sorties audio).

Pour le respect de tous, le bien être et le bon développement de ce logiciel, la formule du *CopyLeft* me semble la plus adaptée à la situation. C'est la formule qui permettra au mieux de faire en sorte que la partie "ensemble on va plus loin" puisse se concrétiser dans les meilleures conditions.

## V- Journal de recherches

Ce chapitre est une sorte de compte rendu des idées qui ont été exploitées, des problèmes qui ont été rencontrés ainsi que des choix qui ont été effectués. Au travers d'exemples et d'illustrations, je montrerai ce que ces idées avaient d'intéressant mais pourquoi certaines d'entre elles n'ont pas été retenues.

### V.1- Au commencement était MOELLER...

Comme je le disais dans la première partie de ce mémoire, dans la version MS-DOS du programme (programmé en *Visual Basic*) qu'utilisait M. BERARD, les lumières étaient contrôlées par du matériel MOELLER (<http://www.moeller.fr/>). De nos jours, MOELLER fabrique des contrôleurs améliorés qui peuvent être pilotés via Ethernet. Ces contrôleurs, dans le cas de M. BERARD sont utilisés pour le déclenchement et l'extinction de projecteurs lumière. Le programme développé en *Visual Basic* n'était non seulement plus capable de piloter les nouveaux contrôleurs mais surtout, il n'était plus compatible avec le nouveau système d'exploitation de Microsoft : Windows XP.

(Sur l'ancienne installation du Vigneroscope, M. BERARD utilisait le matériel suivant :

- dans le PC une carte EPC 335.1 (interface Suconet) avec programme simple fourni par MOELLER
- dans le réseau : 3 stations déportées EM4 201 DX1 avec 3 extensions LE4 116 XD1 par station

Voir les fiches techniques de ce matériel sur le CD-Rom qui contient le logiciel -> DocEnSup.)

La grande question était donc, avant toute chose, de savoir s'il serait possible de piloter ces contrôleurs MOELLER nouvelle génération via Ethernet, grâce à l'environnement de programmation Max-MSP. Un rendez-vous fut alors organisé au CNSMD de Lyon avec un technicien MOELLER particulièrement compétent et agréable. Pour ces tout premiers essais, nous avons utilisé l'objet **Max OpenSoundControl** (OSC) développé par le *Center for New Music and Audio Technologies* (CNMAT)(1).

1- Le CNMAT est un centre multidisciplinaire de recherches de l'université de Berkeley en Californie, département musique. Le but du centre est de fournir une base commune où la musique, la science cognitive, l'informatique, et d'autres disciplines se réunissent pour étudier, inventer, et mettre en application les outils de création pour la composition, l'exécution, et la recherche. Le CNMAT a été fondé dans les années 80 par le compositeur Richard FELCIANO.

Le problème que nous avons alors rencontré était un problème de compatibilité Macintosh – PC. Nous envoyions nos informations au format **OSC** et le contrôleur les attendait au format *User Datagram Protocol* (UDP). En effet, l'objet **OSC** du CNMAT envoie les informations sous un format particulier qui n'est pas exactement le format standard Ethernet. En fait, **OSC** a une mémoire qui lui permet de stocker des messages à envoyer. Ces messages seront envoyés à réception d'un message bang sous forme d'une liste de sous-listes. Il faut donc arriver à envoyer les messages les uns après les autres indépendamment, et non sous forme de sous-listes. Pour cela, il a été prévu un message à envoyer à **OSC**. Ce message est : `resetallthewaymode`. Il est cependant annoté "*at your own risks*" dans la documentation de l'objet. Après quelques recherches, nous communiquions avec succès.

Il existe cependant un autre objet qui permet de ne plus passer par ce formatage **OSC** : **OTUDP**. Cet objet permet d'envoyer les messages Ethernet au format UDP.

Une fois cette question résolue, il fut définitivement décidé que l'application serait développée avec l'environnement Max-MSP et Max-MSP seulement (pas d'interaction avec *Visual Basic*).

Nous n'en étions qu'au début :

- les contrôleurs MOELLER coûtent très cher (ou plutôt les automates qui permettent de contrôler les contacts. Pour chaque automate, il fallait compter environ 600€ et il nous en aurait fallu une grande quantité)
- il fallait encore définir une interface utilisateur
- il restait à mieux définir les fonctions à développer
- voulait-on du son en multi-canal ou en format 6.1 ?
- de quel degré de précision temporelle avions-nous besoin au niveau du séquençement des ordres ? ... etc.

Nous définissions ensemble (M. BERARD et moi-même) une première idée d'interface utilisateur. L'interface se devait d'être très intuitive et donc très simple d'utilisation tout en restant la plus souple possible en termes de configuration.

## **V.2- Définir un type d'interface utilisateur**

Au départ Monsieur BERARD était arrivé avec l'idée de refaire ce qu'il connaissait déjà (un programme plus ou moins similaire). Cependant, après plusieurs discussions et exemples lui expliquant les diverses possibilités de Max-MSP, nous étions vite arrivés à l'idée de ne plus garder que les grandes lignes de départ, à savoir : piloter des réseaux lumières, du son et de la vidéo afin de créer un spectacle.

Pour le reste, plus le temps passait et plus j'ai eu de liberté de proposer de nouvelles idées et/ou solutions. Le programme n'était plus vraiment limité par des restriction mais était devenu de plus en plus ouvert à de nouvelles possibilités.

### V.3- Le modèle du séquenceur

Quand on parle de montage son ou vidéo, bien souvent la première interface qui vient à l'esprit est celle du séquenceur du type *Pro Tools* ou encore *Digital Performer* (1). L'idéal était bien entendu d'avoir une interface de ce type car elles a l'avantage d'être claire, intuitive et donc facile d'utilisation tout en étant puissante et flexible. Max comprend plusieurs objets "interface utilisateur" intéressants à exploiter, je pris donc la décision de creuser dans cette direction.

L'objet **lcd** devait me permettre de dessiner des courbes de contrôle et c'est vers cet objet que se sont orientés mes premières recherches. La première difficulté qui me soit apparue de suite avec ce genre d'interface utilisateur est le fait de ne pouvoir déplacer des objets dans un patcher Max sans que celui-ci soit en mode édition. En effet, normalement, quand un patcher Max n'est pas en mode édition, il n'est pas possible de déplacer les objets qui le composent. Or, je ne pouvais pas demander à l'utilisateur d'ouvrir le patcher (le mettre en mode édition) chaque fois qu'il voudrait déplacer un objet ! Le travail de montage avec un tel système deviendrait vite un enfer et ce serait la porte ouverte aux plus graves erreurs (exemple : effacer un objet primordial par mégarde). Une autre difficulté était d'ajouter des objets dans un patcher qui ne soit pas en mode édition car cette opération est contraire au mode de fonctionnement prévu dans Max-MSP.

Cette dernière difficulté n'en était pas vraiment une en définitive car il a été prévu dans Max une possibilité d'utiliser du script. Avec le script, il est possible de créer des objets, les nommer, changer leur taille, leur couleur, les effacer mais aussi... les déplacer... j'y reviendrai. Je ne vais pas m'attarder sur la méthode d'utilisation du script car les documentations de Max-MSP sont particulièrement bien faites. Si vous voulez en savoir plus sur le script, merci de vous reporter aux tutoriels 46 et 47 du *TutorialsAndTopics.pdf* de Max ainsi qu'à l'objet **thispatcher** dans le *MaxReferenceManual.pdf*.

1- Digital Performer et Pro Tools sont des stations de travail audionumériques et séquenceurs MIDI conçus pour le musicien professionnel. Ils fournissent un environnement complet pour l'enregistrement, le montage, le mixage et le *mastering* d'une grande variété d'applications. Leur vitesse, leur précision et leur flexibilité en font les séquenceurs audio de choix des musiciens professionnels du monde entier. Digital Performer et Pro Tools permettent d'enregistrer et de jouer simultanément de multiples pistes audio et MIDI dans un environnement à la fois créatif et intuitif. Digital Performer ainsi que Pro Tools fournissent un contrôle et une ergonomie exceptionnelle à la réalisation de projets musicaux.

## Voici un premier résultat :

Je ne vais pas trop m'attarder sur ce résultat car il n'est pas très performant. Cela dit, je le propose quand même car il représente l'idée de départ de mes recherches. (Ces patches se trouvent sur le CD-Rom dans le dossier -> BirthPatches -> script\_function\_COMmande.pat et script\_function\_RECEPTION.pat.)

Si vous vous référez à la copie d'écran scriptComAndRec située en annexe dans la partie photo, vous trouverez une vue d'ensemble des patchers. Chaque patcher possède son propre nom. En voici deux :

- script\_function\_COMmande
- script\_function\_RECEPTION

Chacun de ces patchers utilise des lignes de commandes faites en script Max. Le patcher de commande contient six parties :

- créer l'objet
- changer sa longueur
- changer sa coordonnée X
- changer sa durée temporelle (setdomaine (en millisecondes))
- effacer son contenu
- détruire l'objet

Chacune de ces lignes de commande envoie des "ordres" qui sont réceptionnés dans le patcher script\_function\_RECEPTION.

Cette première mise en oeuvre fonctionnait plutôt bien mais les plus gros inconvénients de cette méthode étaient la quantité de paramètres que l'utilisateur devait spécifier ainsi que l'ordre dans lequel ces paramètres devaient être donnés. Même en tant que développeur, il m'arrivait d'essayer d'envoyer un message alors que je n'avais pas encore envoyé celui qui devait le précéder... Cela n'avait pas de conséquence grave mais générait des messages d'erreur qui ne sont jamais agréables et qui, de plus, perturbaient l'utilisateur. En faisant essayer ce genre de prototype à un utilisateur dit "lambda" (une personne n'ayant pas une pratique particulièrement poussée de l'informatique), je me rendais vite compte de la somme de problèmes que ce genre d'interface allait créer.

Le premier problème était le manque d'un côté intuitif. L'utilisateur n'est pas habitué à ce genre de manipulation, il préfère généralement un accès plus direct et interactif avec les objets à manipuler. Le côté contraignant lors du déplacement d'un objet par exemple : devoir donner des coordonnées XY en pixels n'est pas une chose évidente pour qui que ce soit. Il aurait fallu par exemple pouvoir déplacer un objet en cliquant dessus, maintenir la souris enfoncée, la bouger avec l'objet qui aurait suivi cette position, et placer l'objet quand on relâchait la souris !! C'est tout simplement ce qu'on appelle communément en jargon informatique : faire du *drag&drop*. Ce qui semble si évident de nos jours car nous le faisons constamment n'est en fait pas évident à programmer. J'ai eu beau lire et relire la documentation Max, les spécificités de chaque objet pouvant éventuellement m'être d'une quelconque utilité, poser des questions sur la *mailing*

*list* de Max, la seule solution que j'ai trouvée fut d'utiliser un objet externe développé par un utilisateur qui m'a gentiment fait profiter de son travail personnel en langage C.

L'objet que j'ai utilisé s'appelle **posit** et fut créé en 2004 par un développeur indépendant nommé Jasch. L'exemple qui suit montre une version considérablement améliorée en terme d'interactivité avec l'objet et d'interface utilisateur.

### Voici une deuxième version :

(Ce patch se trouve sur le CD-Rom dans le dossier -> BirthPatches -> lcdeplace.pat.) L'objet **mwc** fut quant à lui développé par moi-même en langage C. Il permet d'effacer les messages Max de la *Max Window* lors d'un *debugging* (tests et corrections) de programme par exemple.

Si vous vous référez à la copie d'écran lcdeplace située en annexe dans la partie photo, vous trouverez une vue d'ensemble du patch. Chaque patcher possède son propre nom. En voici trois :

- lcdeplace
- [coord\_test] (qui est un sous-patcher de lcdeplace)
- [function] (aussi sous-patcher de lcdeplace)

Commençons pas le patcher lcdeplace car c'est une ébauche de fenêtre de montage. Dans ce patcher, vous voyez cinq rectangles. Le plus grand est l'objet **lcd**. Les quatre autres sont des objets créés par script dans Max comme nous l'avons vu plus haut. Si vous cliquez sur un de ces quatre derniers rectangles, vous pourrez voir son nom apparaître dans une boîte message en dessous de l'objet **lcd**. C'est grâce à l'objet **posit** qu'il est possible de réaliser cela. En fait, **posit** garde en mémoire le nom de tous les objets nommés à l'intérieur du patcher dans lequel il se trouve. Il ne garde pas que leur nom en mémoire mais également leur position. Quand on clique en un point de l'objet **lcd**, on peut en récupérer les coordonnées cartésiennes. Dans le sous-patcher [coord\_test], je compare constamment l'adresse de mon pointeur de souris avec celles des différents objets créés par script. Si les coordonnées de mon pointeur de souris sont comprises à l'intérieur des dimensions d'un objet nommé, alors Max sait que je suis actuellement en train de cliquer sur l'objet de tel nom. Je peux, de cette manière, choisir précisément l'objet que je veux déplacer sans même avoir à ouvrir le patch. Grâce au script, j'ai pu écrire une ligne de commande qui déplace l'objet sélectionné à l'endroit où se trouve mon pointeur de souris si je la maintiens enfoncée. Le script en question se trouve en bas à gauche du patcher lcdeplace :

```
sprintf script offsetfrom %s lcd_main 0 %i %i
```

Avec les sous-patchers [function] [message] et [lcd], je peux créer de nouveaux objets (**function**, **message** et **lcd**), en modifier la taille et/ou le contenu et enfin, je peux les supprimer.

Si l'on part du principe que le temps se déroule graphiquement gauche à droite dans le rectangle **lcd**. Nous avons dans le patcher lcdeplace une première ébauche d'un séquençement de quatre objets permettant d'envoyer des ordres de contrôle de types

message texte, courbe continue et courbe segmentée en points d'inflexion. Je dis bien graphiquement car les données graphiques contenues dans ces objets ne sont pour l'instant pas stockées dans une mémoire pour une éventuelle relecture. Ceci n'est qu'une expérimentation, une recherche à partir d'une première idée.

#### V.4- Le son

Pour le son, bien entendu, les préoccupations n'étaient pas les mêmes. Les questions les plus importantes me semblaient être :

- quel rapport veut-on créer entre le public et le son ?
- veut-on travailler en stéréophonie, en multi-points ou encore avec un système standardisé comme le 5.1 par exemple ?
- veut-on créer des mouvements sonores dans l'espace (spatialisation) ou plutôt recréer un espace sonore, un espace acoustique ?

En effet, les préoccupations sont totalement différentes suivant qu'on cherche à créer des mouvements sonores dans l'espace (comme on le fait sur un acousmonium en musique acousmatique) ou selon qu'on cherche à attirer l'attention du public en un certain point géographique comme en muséographie par exemple.

En terme d'interface, d'autres questions demeurent :

- veut-on pouvoir visualiser les différents canaux individuellement ou est-ce qu'une forme d'onde globale, résultat d'un *bounce to disk* (copie pré-mixée de toutes les pistes) des sept canaux, pourrait être suffisante ?
- veut-on faire autre chose avec cette forme d'onde que seulement en avoir une représentation graphique ?

Ces questions étaient primordiales pour le développement informatique certes, mais également pour la réalisation du projet. Pour le spectateur, la perception et le rapport qu'il aura au son pendant le spectacle sera totalement différent si le son est projeté en façade (stéréophonie) ou s'il est diffusé en multi-points. Cela peut sembler évident à une personne familière avec le travail du son sur ordinateur mais ce ne sera pas forcément le cas de la personne qui utilisera ce logiciel. Aussi, n'oublions pas que ce logiciel fut d'abord développé pour une demande spécifique et que, dans un premier temps, il devait pouvoir y répondre au mieux. Dans le cas présent, Monsieur BERARD n'était pas du tout familiarisé avec le travail du son, que ce soit avec ou sans informatique.

La première étape fut donc de lui expliquer les différences entre la stéréophonie, le multi-point et les systèmes normalisés comme le 5.1. Explications théoriques mais également pratiques afin qu'il puisse prendre conscience de ce que ces différences impliquent. Il n'était pas primordial à ce stade qu'une décision définitive soit prise. Cependant, il était temps d'y réfléchir sérieusement afin de pouvoir choisir des directions à prendre. Il faut toujours penser plus loin que le besoin immédiat.

Monsieur BERARD en avait bien conscience et il restait très à l'écoute de mes conseils ainsi que de ceux de François ROUX. Selon ma propre opinion, le multi-points était la méthode la plus flexible. Elle permettrait à la fois de faire de la stéréophonie

si besoin est mais également de faire de la spatialisation, de créer des mouvements sonores dans l'espace. Je reviendrai sur ces points car, pour l'instant, nous n'en étions pas là dans le développement du logiciel.

## **V.5- Une interface graphique pour les fichiers audio**

Pour visualiser le son dans MSP, j'utilise l'objet bien connu **waveform~**. Cet objet ne permet pas de d'afficher un fichier multi-canal. Si l'on veut pouvoir afficher tous les canaux avec l'objet **waveform~** il faut tout simplement autant d'objets **waveform~** que de canaux dans le fichier son. Ça prend rapidement beaucoup de place au niveau graphique dans la fenêtre du patcher.

Afin que quelque chose apparaisse dans l'objet **waveform~**, il faut que ce dernier soit lié à l'objet **buffer~**. Ce dernier objet n'accepte pas plus de quatre canaux. Ce qui signifie que le seul moyen d'utiliser plus de quatre canaux avec l'objet **buffer~** est d'utiliser plusieurs objets **buffer~**, un objet **sfplay~** (avec comme argument le nombre de canaux à utiliser), et enfin un autre objet, **poke~**, pour écrire dans les différents **buffer~** ce qui est lu par **sfplay~**. Cette mise en oeuvre est un peu lourde et le problème est que si l'utilisateur utilise un fichier son qui contient moins de canaux que le nombre de canaux utilisés par l'objet **sfplay~**, alors il y aura copie des premiers canaux sur les canaux non utilisés de **sfplay~**. Par exemple, si **sfplay~** attend un fichier son de six canaux et que le fichier chargé n'en contient que quatre, en sortie de **sfplay~** on trouvera :

- canal-1 du fichier -> canal-1 **sfplay~**
- canal-2 du fichier -> canal-2 **sfplay~**
- canal-3 du fichier -> canal-3 **sfplay~**
- canal-4 du fichier -> canal-4 **sfplay~**
- canal-1 du fichier -> canal-5 **sfplay~**
- canal-2 du fichier -> canal-6 **sfplay~**

Et bien entendu, les canaux 5 et 6 de **sfplay~** seront affichés dans **waveform~**.

Ce problème ne se poserait cependant qu'avec d'autres utilisateurs que Monsieur BERARD car celui-ci utiliserait toujours autant de canaux que de haut-parleurs dans son spectacle. Vous me direz « peut-être pourrait-on écrire un script qui permettrait de rectifier ce problème... ». Oui mais dans ce cas là, cela implique de toujours devoir redémarrer Max afin de réinitialiser les DAC (*Digital to Analog Converters*) à chaque changement de nombre de canaux.

En regardant dans les annexes photo, vous trouverez un exemple de visualisation des différents canaux d'un fichier son de sept canaux dans l'objet **waveform~** (photo intitulée multiCanal.jpg). Sur le CD-Rom, dans le dossier BirthPatches, vous trouverez également le patcher MSP qui correspond à cette photo sous le nom : multiCanalWF.pat.

Je proposais cette solution à Monsieur BERARD mais nous en arrivions rapidement à la conclusion que ce n'était décidément pas la meilleure méthode et que le fait de voir les différents canaux n'était pas forcément utile non plus. Pour l'instant, nous



décidions de conserver l'idée d'un fichier son multi-canal. Pour ce qui est de sa représentation graphique, nous en faisons ce qu'on appelle dans le langage du studio un *bounce*. C'est à dire que les sept canaux sont pré-mixés dans le **buffer~** relié à **waveform~** comme un fichier mono. Cela ne nuit en aucun cas à l'écoute car nous sommes bien d'accord que ce n'est que pour la représentation graphique. C'est, pour utiliser un autre terme du studio, une action "non destructive". Pour l'instant, nous en restions là en ce qui concerne les questions sur le son mais je restais convaincu que ce n'était pas la bonne solution. Si les choses restaient telles-queelles pour le projet de Monsieur BERARD, j'en ferais une autre version afin de rendre le logiciel plus flexible et ouvert à d'autres projets.

Vous trouverez une copie d'écran de ce patcher son en annexe photos de ce mémoire ainsi que dans le dossier photo du CD-Rom.

## V.6- Alzheimer chez les objets graphiques

Comme vous l'aurez bien compris, que ce soit l'objet **lcd** ou l'objet **function**, depuis le début de ces recherches, j'utilise des objets Max afin de tracer des courbes de contrôle. Ces courbes de contrôle ont pour but de générer des valeurs qui doivent permettre de contrôler l'intensité d'un projecteur de lumière par exemple ou encore, contrôler le niveau en décibels d'une sortie audio déterminée.

L'objet Max **function** permet de tracer des courbes dites segmentées en points d'inflexion et l'objet **lcd**, dans l'idéal, devrait permettre de tracer des courbes plus souples, sans points d'inflexion.

Pour les courbes de contrôle segmentées, nous avons décidé d'utiliser l'objet Max **function**. Il a l'avantage d'être très flexible dans sa programmation. On peut en changer la durée ainsi que l'ambitus des valeurs en Y. Il possède également un autre avantage d'importance : il mémorise la courbe qu'il contient lors de la sauvegarde du patch dans lequel il se trouve.

Ce n'est pas le cas de l'objet **lcd** qui semble désespérément souffrir d'Alzheimer. Si l'on dessine une courbe dans un objet **lcd** et qu'on sauvegarde le patch, à la prochaine ouverture du patch, les données dessinées dans **lcd** n'auront pas été conservées.

Le problème de la mémoire pour les objets graphiques est quelque chose de lourd à gérer dans tout logiciel. Pour bien faire, il faut que chaque objet ait sa propre mémoire afin de pouvoir se re-dessiner à l'ouverture de l'application. Le problème avec l'objet **lcd** c'est qu'il fut à l'origine développé comme un simple afficheur graphique et non comme objet de contrôle comme **function** ou encore **MultiSlider** par exemple. L'objet **lcd** peut recevoir une grande quantité de messages dont un qui permet, en théorie, à l'utilisateur de récupérer en temps réel les coordonnées cartésiennes de chacun des points de la courbe qu'il est en train de tracer dans l'objet. Le message qui permet d'utiliser cette fonction de l'objet **lcd** est le message **idle** suivi de 1 ou 0 selon qu'on veuille affecter cette fonction à l'objet

**lcd** ou non (1 = oui, 0 = non). Si **lcd** reçoit le message idle 1, alors il enverra les coordonnées cartésiennes du pointeur de souris via sa deuxième sortie à condition que le bouton de la souris soit abaissé et que le pointeur de souris se trouve à l'intérieur de l'objet **lcd**.

Théoriquement, en connectant un objet **capture** à cette deuxième sortie de **lcd**, on devrait pouvoir garder trace de chacune de ces coordonnées car l'objet **capture** est un objet qui sert de mémoire. J'utilise le mot « théoriquement » car, après quelques essais, il se trouve que si le pointeur de souris n'est pas manipulé lentement par l'utilisateur, seule une valeur sur dix ou quinze sera envoyée à l'objet **capture** et de ce fait mémorisée. Cela est dû au faible taux d'échantillonnage de la position de la souris dans **lcd**. La seule façon de récupérer les valeurs de la courbe tracée dans **lcd** serait de la tracer extrêmement lentement.

Bien entendu, en terme d'interface utilisateur, demander de tracer les courbes extrêmement lentement n'est pas possible. En outre, vouloir utiliser l'objet **lcd** pour tracer des courbes de contrôle implique une très grande quantité de données numériques à stocker dans des objets comme **capture**, ou encore **coll**, afin de les récupérer ultérieurement. Il faut imaginer une paire de valeurs numériques représentant les coordonnées cartésiennes de chacun des pixels du tracé de la courbe dans **lcd**. On atteint très vite des listes énormes de valeurs. Sur un seul objet **lcd**, cela peut éventuellement se faire mais si l'on commence à utiliser trente objets **lcd** de cette manière, partant du fait qu'il faut un objet **coll** par **lcd** pour le re-dessin, ça va rapidement créer une application qui risque d'être instable et même assez longue à charger (le temps de re-dessiner les trente **lcd** plus tous les autres objets comme **function** par exemple).

D'autres difficultés sont à prévoir. Si l'on ne veut changer qu'une partie de la courbe, que ce passe-t-il ? C'est possible bien sûr, mais lourd à mettre en place. Si l'on change la durée de l'objet **lcd**, que se passe-t-il ? Il va falloir faire des interpolations qui risquent d'être peu efficaces au moment d'interpréter les données de la courbe. Le but est de programmer une application qui fonctionne bien entendu, mais le but est aussi de faire de la belle programmation. Une programmation lisible, évolutive sans avoir à récrire tout le code, une programmation optimisée et efficace.

Plus le temps passe et plus l'idée d'utiliser l'objet **lcd** pour gérer des courbes de contrôle me semble mauvaise. L'objet **function** permet, de toutes façons, de créer des courbes segmentées qui pourront être interpolées de manière très précise grâce à l'objet MSP **line~**. C'est d'ailleurs la raison pour laquelle **function** avait été développé : en tant qu'interface utilisateur de contrôle pour l'objet MSP **line~**. **Function** combiné à **line~** permet de générer un signal de contrôle continu. **Function** tout seul permet de générer des messages de contrôle par paliers. On peut donc arriver à piloter des projecteurs de lumière en mode « tout ou rien » avec le seul objet **function**.

Après discussions avec Monsieur BERARD et François ROUX, à mon grand soulagement, nous en arrivons donc à abandonner l'idée d'utiliser **lcd** comme source de contrôle. La programmation qui en découlait n'était pas propre et semblait nous conduire dans une application peu évolutive, ce qui n'était pas du tout mon envie. Quant à Monsieur BERARD, en tant qu'utilisateur, **lcd** ne lui semblait pas intuitif dans son utilisation. Or, pour une interface utilisateur, le côté intuitif est très important.

Il ne reste plus désormais qu'un seul objet **lcd** : il sert uniquement à récupérer les coordonnées cartésiennes de mon pointeur de souris dans le patcher afin de déplacer les objets sélectionnés.

Depuis les premières expérimentations, le patch a déjà beaucoup changé. Il est temps de vous en montrer une version globale dont je décrirai certaines parties afin d'expliquer ce qui se passe derrière cette interface.

### V.7- Une nouvelle version

Vous verrez dans l'annexe photos ou encore sur le CD-Rom, dans le dossier photo, une copie d'écran intitulée **interf-proviNew**. Dans le dossier BirthPatches, également sur le CD-rom, vous trouverez le fichier source de ce même patch en extension **.pat**.

Ce que vous voyez là est l'interface utilisateur principale sans aucune connexion cachée. Je l'ai fait exprès afin qu'on puisse mieux imaginer les interrelations entre les objets.

En haut, vous avez l'objet **waveform~** où apparaît la forme d'onde lorsqu'un son est chargé en mémoire (**buffer~**). En dessous, vous avez un axe temporel graduée sur cinq minutes. Pourquoi cinq minutes ? Parce qu'à l'époque Monsieur BERARD pensait ne pas faire de tableau de plus de cinq minutes chacun dans son spectacle. L'idée pour le futur est donc d'avoir un montage par tableau et autant de tableaux qu'on désire par spectacle. Cela dit, comme nous ne sommes pas sûrs de cela pour l'instant, si l'on charge un son de plus de cinq minutes, cet axe temporel s'adaptera à la longueur du son (arrondi à la minute supérieure).

Juste en dessous de cet axe temps, se trouve le fameux banc de montage (**lcd**). C'est là qu'apparaissent les différentes pistes et les objets créés avec leur contenu. C'est aussi là qu'on récupère les coordonnées cartésiennes du pointeur de souris afin de pouvoir déplacer les objets sélectionnés.

Sur l'exemple de la copie d'écran, vous pouvez voir un montage avec quatre pistes créées, une courbe dans un objet **function** et enfin un objet de type message en piste 1.

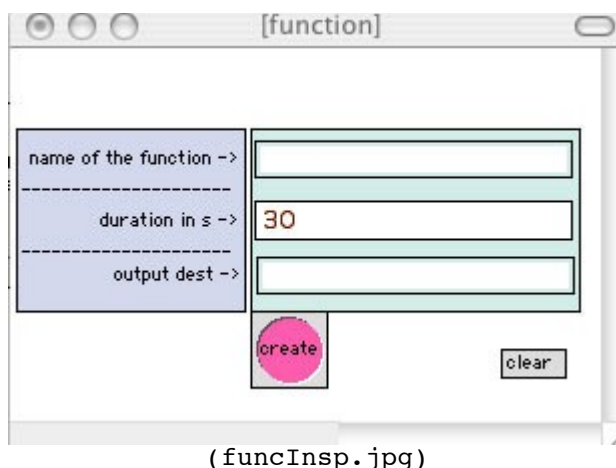
A ce stade de l'application, il n'est toujours pas possible de jouer les données stockées en mémoire dans les **coll**. Pour l'instant, le but est de trouver des idées et de les explorer afin de voir où elles nous mènent. Il nous faut aussi proposer des modèles d'interfaces utilisateur et voir comment une personne étrangère à ce genre de manipulation prend l'application en mains. Encore une fois, il faut que ce soit intuitif. Trop souvent, le développeur, tant il est pris dans son travail, finit par ne plus être en mesure de juger de l'intuitivité de son interface. Il est

absolument nécessaire de faire pratiquer des personnes qui ne soient pas familiarisées avec l'interface afin de voir comment elles s'y prennent, ce qu'elles comprennent et ce qu'elles croient comprendre. Parfois, on découvre que l'utilisateur ne se comporte absolument pas comme on aurait pu l'imaginer et il faut alors impérativement rectifier l'interface. Ici, le premier problème en terme d'interface utilisateur auquel je n'aurais pas pensé est le suivant : il semble difficile aux différents utilisateurs d'avoir à sélectionner un objet avant de pouvoir le déplacer.

Ce qui me semblait évident ne l'est absolument pas pour les différents utilisateurs qui l'ont testé. Ils s'attendaient tous à ce que l'objet suive la souris quand on clique sur l'objet et qu'on déplace la souris. Dans mon système, il faut cliquer sur l'objet pour le sélectionner puis ensuite cliquer ailleurs pour le déplacer à cet endroit. Quand on lâche la souris, c'est cette adresse qui sera enregistrée en mémoire (dans l'objet **coll**). C'est cependant cette interface que je vous propose ici.

### V.7.1 La fenêtre « *inspector* »

Pour ce qui est de l'interface de création des objets, nous avons voulu garder le modèle de la fenêtre « *inspector* ». Non seulement c'est habituel à l'environnement Max-MSP mais ça a le mérite d'être clair, propre et sûr. Voici à quoi elle ressemble.



(Si vous utilisez le patcher interf-proviNew.pat fourni sur le CD-Rom dans le dossier BirthPatches, vous pouvez obtenir cette fenêtre en cliquant sur le menu déroulant situé au dessous de l'objet **waveform~**. Sélectionnez « *function* ».)

Afin d'éviter tout problème de séquençement de données de création lors de la réception par les scripts, le processus doit se faire en quatre temps :

- *name of the function* (nom de la fonction) c'est le nom de l'objet **function** créé (ce nom n'apparaîtra pas dans le **lcd** mais il permet à Max d'identifier les différents objets créés)
- *duration in s* (durée en seconde) -> par défaut = 30 sec.
- *output dest* (sortie) sera un canal MIDI ou Ethernet... à voir
- *create* (un clic sur ce bouton réalisera la création)

Ce n'est donc qu'après avoir cliqué sur le bouton *create* que l'objet sera enregistré en mémoire. Ce bouton permet d'envoyer les données de création dans l'ordre attendu par les scripts Max et ceci indépendamment de l'ordre dans lequel l'utilisateur les aurait spécifiées.

Je ne montrerai pas toutes les différentes étapes par lesquelles passent ces données de création car je ne voudrais pas passer trop de temps sur ce patcher. Pour votre information, voici les scripts Max de création qui reçoivent les données et donc l'ordre dans lequel celles-ci doivent leur parvenir :

```
sprintf script new %s user function 266 210 378 260 1 1 0 0  
(création du nouvel objet function de nom x, à telle adresse dans la fenêtre du lcd)
```

```
sprintf script size %s %i 50  
(donne la taille n à l'objet de nom x)
```

```
sprintf script send %s setdomain %f  
(donne la durée n à l'objet de nom x)
```

Les données sont ensuite concaténées afin de ne former plus qu'une seule liste de la façon suivante :

pack s i f s, où

- s = symbole
- i = entier (*integer*)
- f = flottant
- s = symbole

La liste résultante aura donc la forme :

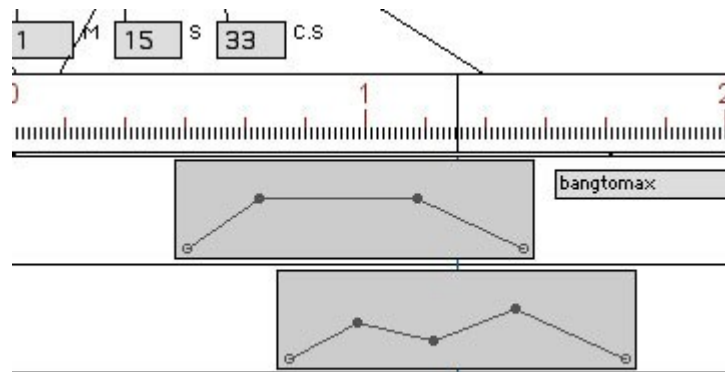
- nom taille durée sortie

C'est sous cette forme que seront stockées en mémoire les informations des types précités relatives à chaque objet **function** de nom x.

Pour l'objet **message**, le processus est sensiblement identique. La durée doit pouvoir changer afin qu'on puisse définir le début et la fin d'une action en fonction de la longueur de l'objet dans la fenêtre de montage. Dans cette version, le réglage de la taille fut laissé au stade de développement car nous sommes ensuite partis sur d'autres idées que j'expliquerai plus tard quand je parlerai du patch actuel. Le tout, à ce stade, est que les objets **message** puissent être de dimension suffisante que pour contenir un message explicite pour l'utilisateur.

### V.7.2 Le pointeur de lecture

Un pointeur de lecture dans un séquenceur est une chose indispensable.



(pointeur.jpg)

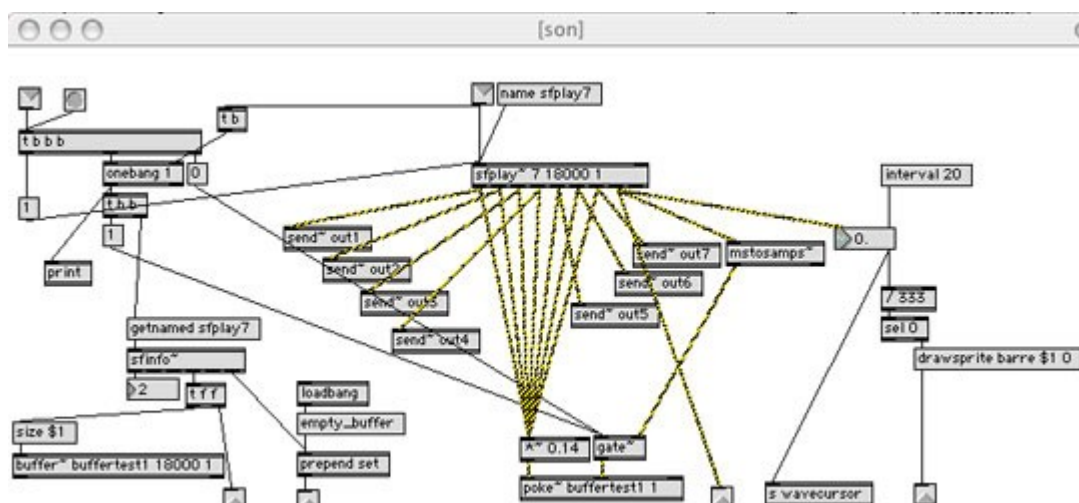
Dans cette application, c'est le son qui pilote tout, c'est lui qui donne l'horloge de séquençement des évènements. A ce stade, il ne s'agit que d'une question graphique mais dans les versions futures, les différents évènements programmés dans le banc de montage prendront effet suivant la position du pointeur de lecture dans le fichier son.

Tout d'abord, voici le message Max qui permet le dessin du pointeur de lecture de nom « barre » dans le banc de montage :

```
recordsprite, paintrect 0 0 1 10000 200, closesprite barre,
drawsprite barre 0 0
```

Avec ce message, je crée le pointeur de lecture, je lui donne une dimension et une couleur ainsi qu'un emplacement.

### V.7.3- Patcher son et pointeur de lecture



(SoundPatchbeg.jpg)

Ce patcher, même s'il sera amené à subir des modifications dans les versions à venir, comporte déjà beaucoup d'éléments qui resteront. Le son est lu en *direct to disk* par **sfplay~**. Cela permet de pouvoir jouer un son très long sans avoir à le charger

préalablement dans une mémoire **buffer~** (cela pourrait prendre un temps relativement long suivant la longueur du fichier son à charger). Il est également possible de lire un fichier son comprenant plus de quatre canaux sans aucun problème.

L'objet **sfplay~** possède une sortie qui donne, sous forme de signal, la position de son pointeur de lecture en millisecondes. Ce sont ces données que je récupère et traduis en pixels afin de dire à Max de positionner le pointeur de lecture sur tel pixel puis sur tel pixel et ainsi de suite durant toute la lecture du fichier son. Voici le message Max qui permet cela et que vous pouvez trouver à droite, dans la copie d'écran précédente :

```
drawsprite barre $1 0
```

\$1 étant la variable « adresse X » du pointeur de lecture.

Quant à la relecture ultérieure des données stockées en mémoire dans les **coll**, la question se pose encore de savoir comment faire au mieux. Faut-il regarder continuellement si, à la position du pointeur dans **lcd**, correspond un déclenchement à effectuer ? Ou, peut-être plus performant : faut-il utiliser l'adresse du premier objet comme point de référence d'une sorte de minuterie dans laquelle chaque déclenchement suivant serait calculé comme un délai du temps initial ?

La deuxième solution offre l'avantage d'éviter d'être constamment en train de comparer la position du pointeur avec les différentes adresses enregistrées dans les différentes **coll**. Logiquement, c'est moins de travail pour l'ordinateur et donc, une programmation plus optimale... à voir. Ce n'est pas le plus important : convertir une valeur en une autre n'est généralement pas bien compliqué. La question reste ouverte pour l'instant. Si la deuxième solution venait à être choisie, le pointeur de lecture dans le banc de montage ne servirait alors que de repère visuel pour l'utilisateur.

#### V.7.4 La mémoire -> coll.

A ce stade, le point sur lequel je voulais travailler était la mise en mémoire. En particulier : comment modifier uniquement un seul paramètre sans récrire l'ensemble des données adjacentes ? Pour plus de clarté, j'ai opté pour l'utilisation de plusieurs **coll**. Un pour chaque paramètre, le tout divisé en deux classes d'objets :

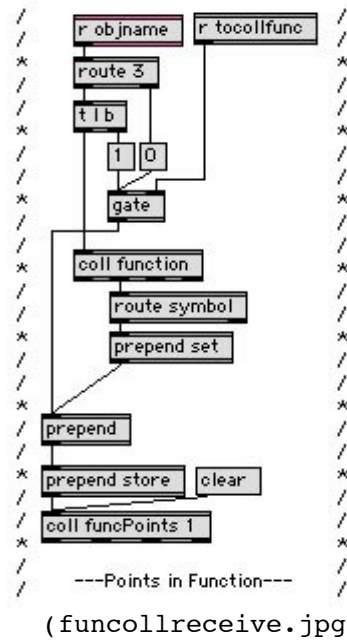
- *function*
- message

Pour la classe *function*, j'utilise trois **coll** :

- général (nom, taille, durée, sortie)
- adresse en X (qui servira éventuellement de délai de relecture)
- adresse des points dans l'objet **function**

Quand on sélectionne un objet, le programme va analyser si l'objet sélectionné est de type *function* ou message. Si celui-ci existe, il sera localisé en mémoire et les anciennes données y correspondant seront tout simplement remplacées par des nouvelles. L'inconvénient d'utiliser cette méthode est que pour chaque **coll**,

j'ai besoin d'un objet **receive** objname. L'objet **receive** objname me permet de récupérer le nom de l'objet que je viens de sélectionner et de l'envoyer dans la mémoire (**coll**) pour le rechercher. Voici un exemple :



Au moment de la création de chacun des objets, le nom que l'utilisateur lui donne est précédé du chiffre 1 ou du chiffre 3 suivant la classe dans laquelle il se situe :

- classe 1 = message
- classe 3 = *function*

Ce patcher analyse chaque nom d'objet sélectionné. Si le nom de cet objet est précédé de 1 alors il sait que l'utilisateur vient de sélectionner un objet de type (classe) message. Dans le cas ci-dessus (ici il attend un objet de type *function*) il ne laisse pas passer l'information (voir **route** 3 dans la copie d'écran). Si le nom de l'objet sélectionné est précédé du chiffre 3 alors il sait que l'utilisateur vient de sélectionner un objet de type (classe) *function*. Dans ce cas, il laisse passer l'information (le nom de l'objet sélectionné). Il regarde dans sa mémoire si l'objet existe déjà et si c'est le cas, il concatène au nom de cet objet la nouvelle liste de valeurs (d'informations) qui y sont attachées.

Cette méthode permet, d'une certaine façon, de diviser par deux le temps de recherche dans les différentes mémoires car si l'utilisateur sélectionne un objet de classe *function* par exemple, seules les mémoire consacrées à cette classe seront analysées.

Cette technique de « classes » m'est venue d'un autre point du patcher : les modes.



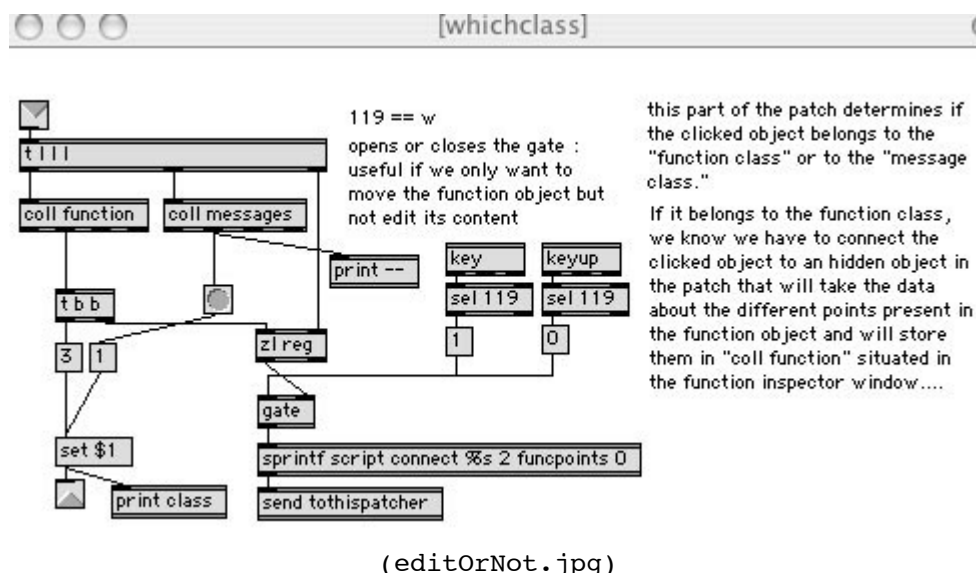
### V.7.5 Les modes

Je me suis rapidement rendu compte qu'il fallait absolument avoir deux modes différents dans ce genre de programmation :

- un mode création
- un mode édition et/ou modification

En effet, pour pouvoir récupérer les valeurs correspondant aux différents points placés dans l'objet **function**, je dois créer une connexion entre la sortie *dump* de l'objet **function** (la troisième sortie en partant de la gauche) et l'objet **coll** qui mémorise ces valeurs. Cependant, cette connexion ne doit se produire que si l'on insère des points dans **function** ou les modifie. Si l'utilisateur ne désire que déplacer l'objet sans en modifier le contenu, il n'y a pas besoin de remplacer les valeurs en mémoire par les mêmes valeurs. De plus, cela pourrait même générer des problèmes dans le cas où les valeurs précédemment présentes en mémoire seraient remplacées par de fausses valeurs : des valeurs d'un autre objet **function** qui seraient restées stockées dans un objet **gate** par exemple.

Voici le programme qui gère cela dans le patcher :



Pour distinguer le mode déplacement du mode édition/modification, j'ai recours à un procédé que j'utilise souvent dans mes programmes : la combinaison de touches (ou le raccourci clavier). Si l'objet est sélectionné avec la touche **w** du clavier enfoncée (code ASCII 119), le programme est en mode édition, sinon il est en mode déplacement. Ce programme détermine si l'objet sélectionné par l'utilisateur appartient à la classe *function* ou à la classe *message*. Comme vous pouvez le voir dans la photo ci-dessus, il y a deux objets **coll** :

- **coll function** contient le nom de tous les objets existant de classe *function* dans le banc de montage
- **coll messages** fait de même avec les objets existant de classe *message* dans le banc de montage



patcher et on le sauvegarde avec l'extension .pat (format texte de préférence). Fermer le patcher signifie que celui-ci n'est plus en mode édition et que la barre d'outils n'est plus visible.

Maintenant que les termes « ouvert » et « fermé » sont définis, voici le problème que je rencontre systématiquement. En mode ouvert, déplacer un objet au-delà de la taille de la fenêtre (du patcher) n'est pas un problème. Il suffit de faire du *drag&drop* avec l'objet et de le placer où l'on veut dans la fenêtre. Si l'objet est placé au-delà des limites actuelles de la fenêtre, pas de problème, des ascenseurs apparaissent sur les côtés et permettent ainsi à l'utilisateur de se déplacer dans la fenêtre quelle qu'en soit la taille qui est donc augmentée automatiquement. En mode fermé, par contre, en utilisant le script, les choses ne se passent pas du tout de la même façon. Essayez de déplacer un objet au-delà des limites actuelles de votre patcher à l'aide du script et vous verrez que les ascenseurs n'apparaîtront pas ! C'est très agaçant.

Les seules solutions que j'ai trouvées pour contourner ce problème ne sont en fait pas vraiment des solutions mais plutôt des *hacks* (des astuces). Je reviendrai sur ces fameux *hacks* plus tard car j'en utilise un dans la version finale du programme. Le problème semble compréhensible en théorie : il semble que ce soit le rafraîchissement graphique de la fenêtre qui se fasse mal quand le patcher est en mode fermé. J'ai signalé le problème à Cycling'74 mais, bien que mon *hack* semble les avoir convaincus, ils en restent là pour l'instant...

#### V.7.6 Bilan de cette version

« Y-a de l'idée » voilà qui pourrait traduire mon sentiment. Certaines fonctions de ce programme sont intéressantes comme le fait de pouvoir déplacer un objet où l'on veut dans un patcher fermé par simple *drag&drop*. L'idée du mode déplacement et du mode édition/modification est aussi intéressante. La fenêtre *inspector* est aussi une idée à creuser. Mais plus le temps passe et plus je me dis que ce sont bien là les trois seules choses que je garderais éventuellement de ce programme.

Il y a de multiples fonctions qui restent à déterminer, à modifier ou à mettre en place :

- la mise en mémoire n'est pas encore convaincante car elle utilise trop d'objets **coll**. Finalement, le fait d'avoir un seul et même **coll** qui contienne toutes les informations pour rejouer le montage semblerait la solution la plus propre et la plus stable en termes de programmation. Tout figurerait ainsi dans une seule mémoire
- l'interface n'est pas assez intuitive. Même si pour moi, développeur, elle semble claire, elle ne l'est pas pour un utilisateur lambda
- l'idée à développer est que l'utilisateur effectue ses sélections dans la fenêtre de son (**waveform~**). La longueur et l'emplacement de la sélection détermineront la position de l'objet en X lors de sa création dans le banc de montage, sa longueur et donc sa durée. L'utilisateur n'aura plus qu'à placer l'objet en Y

- la fenêtre *inspector* est une bonne idée, il faut la garder mais la développer
- le fait que chaque objet possède sa propre mémoire de taille, d'adresse et de re-dessin est certainement la solution la plus claire
- il serait bien que pour éditer le contenu d'un objet déjà existant, l'utilisateur n'ait qu'à double-cliquer dessus et y apporter les modifications souhaitées
- le banc de montage type séquenceur est à garder, il ne peut difficilement en être autrement, c'est culturel. Cela dit, le fait d'avoir tout en une seule fenêtre n'est pas ce qui s'annonce des plus évidents. Dédier une fenêtre à une fonction particulière permettrait d'éclaircir l'interface et éventuellement de la rendre plus intuitive

Voilà les principaux points sur lesquels il va falloir travailler dans les versions à venir. Dans l'ensemble, le patcher utilise trop d'objets. Il est peu clair et semble devenir de moins en moins évolutif au fur et à mesure que le temps passe. Bien sûr, avec Max-MSP, il est toujours possible d'ajouter des objets. Le programme finira par fonctionner après un clic ici, un clic là puis un autre ici... etc. Dans le but de dépanner un patcher deux minutes avant un concert, ce genre de programmation est acceptable car c'est du provisoire. Dans un cas comme celui-ci où le but est d'avoir une application stable, évolutive et puissante tout en restant intuitive... il faut penser la programmation différemment.

Les jours passaient, le programme avançait mais François ROUX et moi-même voyions clairement qu'à un moment ou à un autre, nous allions nous retrouver bloqués en persistant dans cette direction. Pour arriver à nos fins, nous devons constamment aller à contre courant des concepts de base de Max-MSP. Le code commençait à devenir sérieusement illisible et nous n'en étions qu'au début du projet !

### V.7.7 L'objet timeline

Le lecteur pourra se demander depuis un certain temps déjà : « mais pourquoi ne pas utiliser l'objet **timeline** ? » A dire vrai, c'est même la première idée qui me soit venue. J'avais beaucoup étudié le patcher d'aide de l'objet et j'avais également lu toute la documentation que j'avais pu trouver concernant **timeline**. En théorie, cet objet est terriblement intéressant. Malheureusement, il est aussi connu pour comporter quelques *bugs* (défaut ; on dit aussi « bogues » en français). J'ai quand même décidé de faire des essais mais deux minutes après, j'avais déjà des problèmes qui n'annonçaient rien de très encourageant, comme par exemple :

- un problème de rafraîchissement graphique qui, quand on déplace un objet dans **timeline**, crée des sortes de traces grisâtres qui gâchent tout l'aspect esthétique
- la taille de l'objet par défaut ne permet pas d'avoir des ascenseurs ; quand on déplace un objet, on récupère donc parfois le problème que nous avions déjà avec le script à savoir un objet placé au-delà de limites graphiques de **timeline** et aucun ascenseur disponible

Malgré ces quelques petits problèmes, l'objet **timeline** avait quand même soulevé des idées qui nous parurent intéressantes à exploiter comme par exemple le fait que, dans **timeline**, ce ne sont pas des objets qu'on charge mais des patchers. On utilise ensuite des objets de type interface graphique qui communiquent avec les patchers chargés. Voilà une idée intéressante à développer. L'idée n'est pas de reprendre **timeline** mais de s'en inspirer. Max offre la possibilité de faire des abstractions (des patchers qu'on appelle dans une boîte objet). Pourquoi ne pas partir sur un principe de *plug-ins* (insérable) ? Chaque *plug-in* aurait une fonction particulière et serait entièrement paramétrable. Au lieu de positionner des objets dans le banc de montage, l'utilisateur positionnerait différents *plug-ins* qu'il pourrait paramétrer selon ses besoins. L'idée nous semblait plus qu'intéressante car elle permettrait de simplifier de beaucoup les questions de mémoire. Aussi bien les mémoires d'exécution du montage que les mémoires propres à chaque objet.

Enfin, **timeline** offre une autre fonction intéressante : la possibilité de positionner des marqueurs à des endroits clef. Ces marqueurs permettent à l'utilisateur d'aller directement à un endroit donné, dans le banc de montage, simplement en appelant le marqueur désiré. Le programme avait besoin d'une telle fonction.

#### V.7.8 L'apocalypse selon timeline

Toutes ces nouvelles idées nous donnèrent l'envie de presque tout reprendre à zéro. Non, nous n'avions pas travaillé en vain, mais nous avons besoin d'un nouveau départ. Comme dit l'expression : « reculer pour mieux sauter ». Cette fois-ci, nous sentions que nous étions enfin sur la bonne voie.

#### V.8 Vers une version évolutive pour *plug-ins*

Quinze jours plus tard, une sérieuse refonte de l'application voyait le jour. Beaucoup de choses avaient changé et certaines d'entre elles fonctionnaient déjà :

- l'apparence générale de l'application était radicalement nouvelle
- l'application possédait maintenant de multiples fenêtres très faciles à ouvrir et fermer pour l'utilisateur
- l'idée du *plug-in* avait été retenue et nous appelions maintenant des abstractions au lieu de créer des objets
- la taille et l'emplacement du *plug-in* dépendait maintenant de la sélection faite dans la fenêtre où apparaissait le fichier son. Même si elle fonctionnait alors plutôt mal, avec du travail ça devait pouvoir s'améliorer
- nous ne parlions plus de jouer de fichier son multicanal. Dorénavant, le fichier chargé serait un fichier monophonique ou stéréophonique qui serait réparti sur toutes les sorties audio. Premièrement, c'est beaucoup plus flexible si un même spectacle vient à être joué dans plusieurs salles dont l'acoustique sera forcément différente
- Deuxièmement, chaque affectation de sortie peut être modifiée à tout moment, ce qui est un grand avantage pour l'utilisateur
- l'objet **lcd** ne servait plus qu'au dessin de l'axe temporel

- l'objet externe **posit** ne faisait plus partie du programme, ce qui nous libérait des éventuels problèmes ultérieurs de portage du code pour les versions à venir de Max-MSP dans le cas où le développeur de l'objet (Jash) n'effectuerait pas lui-même les mises à jour nécessaires

Et enfin,

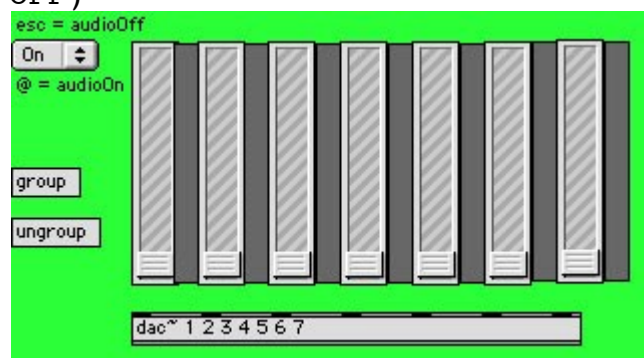
- quatre *plug-ins* étaient mis à disposition de l'utilisateur (contact, vidéo, gradateur, volume) :  
 Contact = déclenchement de type « tout ou rien »  
 Vidéo = projection de vidéo  
 Gradateur = contrôle continu pour les lumières  
 Volume = contrôle des niveaux pour chaque sortie sonore

L'interface utilisateur ayant tellement changé, je vous invite à aller voir la copie d'écran en annexe photos : demo060325. Vous pouvez également ouvrir ce patch dans le dossier BirthPatches du CD-Rom sous le nom V\_060325.PAT. La version V\_060325 n'est en fait pas la première version sur laquelle nous avons travaillé mais c'est la plus ancienne qu'il me reste avec cette nouvelle refonte du patch. La version présentée ci-après n'est cependant seulement le fruit que de quelques jours supplémentaires de travail et ressemble de beaucoup à la toute première nouvelle version.

Pour ce qui est de l'apparence générale de l'interface, la photo parle d'elle-même. Comme vous le savez, chaque patcher possède son propre nom. Ici, SoundWindow, Banc, dac et création sont tous des sous-patchers du patcher principal : V\_060325. Ils s'ouvrent par défaut et se ferment exactement comme n'importe quelle fenêtre sous OS X ou Windows version N. Une fois fermées, ces fenêtre peuvent être ré-ouvertes en double cliquant sur les patchers de noms précités dans la fenêtre V\_060325.

Comme dans les versions précédentes, le son est l'élément maître du patcher. C'est lui qui donne l'horloge de séquencement. Il faut donc commencer par charger un son. Pour ce faire, voici rapidement comment il faut procéder :

- dans la fenêtre dac, positionnez le menu sur ON afin de mettre l'audio en marche (un raccourci clavier a également été prévu @ = On, esc = OFF)



(dacvert.jpg)

- il vous faut ensuite choisir le son à charger en cliquant sur open dans la fenêtre SoundWindow

- une fenêtre de dialogue apparaît et vous permet de choisir un son dans l'arborescence de votre ordinateur. Une fois celui-ci choisi, cliquez sur *play* toujours dans la fenêtre SoundWindow et laissez le son jouer jusqu'au bout. Ne vous inquiétez pas si la vitesse de lecture est trop rapide. Tout va bien : une fonction permet de charger le son en mémoire plus rapidement et cet effet ne se reproduira plus lors des lectures ultérieures

Je reviendrai plus en détail sur certains points comme la construction de ces différents patchers. Cependant, je ne le ferai que dans le chapitre consacré à la version finale de l'application. Le but ici est seulement d'expliquer les grandes lignes et non les détails de la programmation. Dans cette version de l'application, seul le *plug-in* contact fonctionne plus ou moins (plutôt moins que plus d'ailleurs).

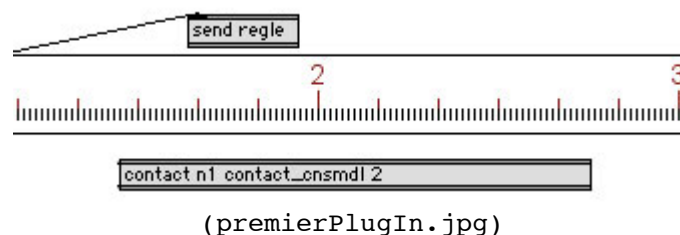
Pour obtenir le résultat de la copie d'écran suivante, voici rapidement comment insérer un *plug-in* contact dans le banc de montage :

- dans la fenêtre creation, cliquez sur open et écrivez un bref commentaire comme par exemple contact\_cnsmdl. Fermez cette fenêtre
- dans le menu déroulant, toujours dans la fenêtre creation, choisissez contact
- enfin, entrez un nombre dans l'objet **number box** (ou encore boîte nombre) à gauche de ce même menu. Exemple 1. Validez.

Plusieurs choses apparaissent alors :

- l'objet contact apparaît dans le banc de montage (un peu n'importe où, il est vrai). Si vous avez entré le nombre 1 comme moi, votre objet contact s'appelle n1
- dans la fenêtre SoundWindow, vous trouverez un menu déroulant à côté du nom du fichier son que vous aurez chargé. Dans ce menu devrait apparaître votre objet contact sous le nom n suivi du nombre que vous avez entré (dans mon cas : n1)
- pour déplacer cet objet en X et Y, ou encore le redimensionner, il vous faut le sélectionner dans ce menu déroulant
- la boîte nombre à droite de ce menu déroulant donne l'adresse Y de l'objet, vous pouvez y mettre 117 par exemple
- effectuez maintenant une brève sélection dans l'objet **waveform~** (de préférence en tout début de fichier) et votre objet se placera en X. Sa taille sera totalement dépendante de la longueur de votre sélection

Si tout va bien, vous devriez avoir quelque chose comme cela dans votre banc de montage :







En haut à droite de cette copie d'écran, vous pouvez voir trois objets différents : **key**, **keyup** et **sel** 115. Il s'agit encore une fois d'un raccourci clavier. Plus haut, je disais qu'il fallait un système de pointeur qui ressemblerait un peu à ce qui a été fait dans l'objet **timeline**. Voici à quoi sert cette fonction : une fois que vous avez chargé un son et que vous avez des objets placés dans votre banc de montage, vous serez forcément amené à revenir plusieurs fois à un certain point de votre montage afin de le tester jusqu'à ce qu'il vous convienne. Si vous maintenez la touche « s » du clavier enfoncée alors que vous cliquez à un endroit dans la barre de temps du banc de montage, la lecture du fichier son commencera à cet endroit précis. Ceci vous permet de travailler sur une partie déterminée de votre montage. Pourquoi la lettre s ? Car le message utilisé pour faire cela avec **sfplay~** est le message `seek $1` où `$1` est la variable temps du pointeur de lecture de **sfplay~**.

## V.9 Bilan de la version 060325 et orientation vers une version finale

Malgré les nombreux *bugs* de cette version, nous étions convaincus d'avoir pris la bonne décision en apportant tous ces changements. L'application y gagnait fortement en lisibilité et en facilité d'utilisation. Nous pouvions déjà envisager de la faire beaucoup évoluer sans devoir nous avancer sur des propositions irréalisables.

Les points principaux sur lesquels il restait à travailler étaient les suivants :

- la sélection dans le fichier son doit correspondre à l'emplacement ainsi qu'à la taille de l'objet édité dans le banc de montage
- le *plug-in* doit avoir une taille graphique minimale afin que l'utilisateur puisse garder un minimum d'informations visibles à l'écran. Je dis bien graphiquement car les données stockées en mémoire, elles, doivent impérativement correspondre à la taille de la sélection. Pour l'instant, une sélection minuscule finit par faire disparaître l'objet du banc de montage
- quand l'utilisateur double-clique sur un objet déjà existant afin d'en éditer le contenu, la sélection correspondante dans le fichier son doit être visible dans la SoundWindow (processus inverse)
- il serait bien d'avoir des fonctions **zoom** dans l'objet **waveform~** pour le fichier son mais également dans le banc de montage
- il faut travailler également sur les autres *plug-ins* qui pour l'instant ne sont pas fonctionnels. Notamment le *plug-in* vidéo que nous voulions créer sans utiliser Jitter (afin d'éviter aux futurs utilisateurs d'avoir à dépenser \$395 alors que le patch n'utiliserait que deux objets Jitter tout au plus)
- enfin, il faut penser à la relecture (l'interprétation) car cette fonction n'a pas été traitée pour l'instant

Une fois toutes ces modifications apportées, nous en arriverions à une version quasi finale où seules quelques améliorations seraient nécessaires mais où l'essentiel serait présent.

Par version finale, je fais référence au projet Vignéroscope bien entendu car grâce à ce séquenceur multimédia, Monsieur BERARD sera en mesure de faire plus que ce qu'il imaginait quand il s'est adressé au CNSMD de Lyon en début d'année. Le logiciel quant à lui, est amené à être développé et amélioré pendant quelques temps encore car finalement, il n'y a pas vraiment de limite à ses possibilités et c'est ce qui en fera un excellent logiciel dans le futur.

Les différences entre la version 060325 et la version actuelle du patch ne sont finalement que de longues corrections de mille et un *bugs*, et le contenu de ce mémoire se doit de tenir dans un certain nombre de pages. Je vous épargnerai donc toutes les explications propres à la résolution de ces *bugs*. Explications qui ne seraient que trop techniques et pourraient même devenir laborieuses pour le lecteur. Bien entendu, je suis à votre disposition si vous désirez de plus amples informations sur un point qui vous intéresserait particulièrement. (Voir mon adresse courriel en avant-propos de ce mémoire.)

## VI- Renseignements techniques concernant les périphériques externes

L'application finale, telle qu'elle sera décrite dans le chapitre VII de ce mémoire, utilise le réseau Ethernet pour communiquer avec une carte externe : la carte GLUION. C'est cette carte externe qui va recevoir nos messages de contrôle au format UDP pour tout ce qui concerne les lumières, que ce soit en contrôle « tout ou rien » ou en contrôle continu. Carte GLUION, Ethernet ou encore UDP et bien d'autres, tous ces termes sont expliqués dans ce chapitre six.

### VI.1- La carte GLUION

Au chapitre V.1, nous avons vu que l'entreprise MOELLER propose des socles pré-programmés qui permettent le contrôle par Ethernet des contacts de même marque qui y ont été insérés. Les contacts sont comme des interrupteurs qui fonctionnent par l'alimentation électrique d'une bobine interne. L'avantage de ce matériel MOELLER est la robustesse, il résistera certainement à un taux d'humidité élevé par exemple. Son inconvénient est son coût. En effet, pour un socle n'ayant qu'une dizaine de sorties possibles, il fallait compter dans les 500€ au minimum alors que nous avons besoin d'une trentaine de sorties. De plus, ces boîtiers sont relativement imposants comparés à la carte GLUION. Enfin, il ne permettent rien d'autre que le contrôle de contacts MOELLER.

Principalement pour des raisons économiques, nous avons donc cherché d'autres moyens de communiquer avec les contrôleurs par Ethernet et, après quelques comparaisons, nous nous sommes rapidement fixés sur la carte GLUION que voici :



(B\_barefoot.jpg)

Cette carte est fabriquée par Monsieur Sukandar KARTADINATA. De nombreux renseignements techniques sont disponibles à <http://www.glui.de/prod/gluion/gluionManual.html>

La carte GLUION est une nouvelle interface pour capteurs destinée à la musique ainsi qu'aux multimédia et qui se distingue par sa rapidité et sa flexibilité. Son prix est également très attractif : 455€.

## VI.2- Explications techniques et électroniques pour l'utilisation de la carte GLUION

La carte est alimentée entre 6V et 12V maximum, transformé en un niveau de référence interne de 3.3V. Chacune des broches d'entrées/sorties de la carte fonctionne directement sous cette tension maximum de 3.3V, jusqu'à 180 milliampères maximum disponible. Aucun étage d'amplification ni aucune protection quelconque (retours de courant, blindage électrostatique, ...) ne sont implémentés. Travaillant donc en « direct » avec les éléments de la carte sensibles électriquement, nous avons donc dû réaliser une partie électronique pour à la fois :

- isoler les éléments de la carte, sensibles électriquement, des courants amplifiés que nous devons nécessairement utiliser, étant donné qu'une partie du matériel externe à la carte avait déjà été achetée
- monter un étage d'amplification, les signaux de commande devant se hisser au 24V sous 100 MA
- assurer les continuités de la masse, problème généralement délicat sur les cartes de commande/acquisition
- assurer des charges inductives relativement élevées

La carte compte 64 broches numériques et 24 broches analogiques. Chacune des broches peut être programmée soit comme entrée soit comme sortie. Cet état d'entrée ou de sortie est lui-même associé à une fonction de traitement de données spécifique permettant ainsi des types de broches différents, tels par exemple : binaire (1 ou 0), signaux spéciaux comme le *Pitch Width Modulation* (PWM) ou le DMX. La surface totale hors tout est d'environ 13x10 cm pour 64 + 24 voies de commande/acquisition. Physiquement, la densité des composants sur la carte ainsi que la taille des pistes électriques, imposée par le composant principal, est importante. L'utilisation de la carte, pour les essais, nécessite donc une vigilance accrue : géographiquement on peut toucher aisément à la fois le blindage de la prise Ethernet et certaines des broches, lors du branchement Ethernet. Ce qui a pour conséquence de mettre la masse au potentiel de la broche, détruisant ainsi définitivement les composants.

Le coeur même du calculateur est appelé FPGA. Jusqu'à peu on utilisait encore, par exemple, le 8051 qui avait jusqu'à 16K de mémoire et se programmait extérieurement à la carte en plaçant le composant sur un petit boîtier que l'on appelait « programmeur d'EPROM » et auquel on envoyait des programmes pré-compilés par un ordinateur. Un développeur informatique écrivait le programme dans un langage particulier puis envoyait ce programme à l'EPROM via le port série de l'ordinateur. A l'intérieur de l'EPROM, pour la programmer, il fallait commencer par ouvrir l'équivalent de portes. Le programme était envoyé à celles-ci puis, une fois le programme envoyé dans sa totalité, on refermait les portes afin d'en assurer la sauvegarde. Le FPGA est un composant moderne beaucoup plus sophistiqué, plus richement doté en mémoire, etc. Il se programme avec des langages plus généraux (C).

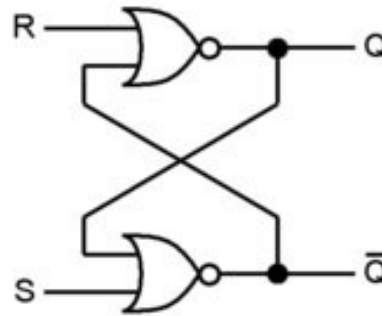
Les composants de la carte sont dit de type CMS (Composants Montés en Surface), c'est à dire qu'ils sont soudés directement sur la carte. Ce sont des composants qui n'ont pas de pattes. Les composants sont posés sur la carte qui elle-même est ensuite passée au four. La technique consiste à poser la soudure avant le composant, ensuite on ajoute le composant sur la soudure et on passe le tout au four à une température de 260°. La soudure fond aux pieds des composants et les scelle à la carte. Les composants électroniques des générations précédentes (dits 'traditionnels' ou 'traversants') étaient d'assez grosse taille et équipés de broches destinées à traverser le circuit imprimé, la soudure se faisant du coté opposé de la carte afin de relier électriquement les broches au circuit imprimé. La miniaturisation constante des cartes électroniques a rendu ce système quasi obsolète :

- les composants sont plus petits et plus légers
- les circuits imprimés n'ont plus à être percés
- l'assemblage peut être automatisé facilement
- les tensions de surfaces recentrent les composants automatiquement sur leurs plages lors de l'étape de brasage. Les marges de placement sont ainsi augmentées
- des composants peuvent être placés sur les deux faces de la carte
- les résistances et inductances électriques sont diminuées, augmentant ainsi les performances en hautes fréquences
- les propriétés mécaniques en vibrations sont augmentées
- le coût global est diminué

Le seul inconvénient se situe au niveau de la maintenance, posant des problèmes supplémentaires aux techniciens assurant le dépannage, particulièrement lorsqu'ils doivent changer un composant.

Il est possible de tirer jusqu'à 100 milliampères par broche, ce qui est relativement important. Selon ce qui était expliqué dans la documentation de la carte, nous avons longtemps pensé que les sorties binaires envoyaient simplement une impulsion de 50 nanosecondes, ce qui est un standard (une nanoseconde équivaut à un milliardième de seconde), et qu'ensuite elles étaient relâchées. Cela aurait nécessité de faire un circuit électronique, entre le projecteur et la carte, qui aurait mémorisé cet état ouvert (le bit 1) le temps désiré et qui serait repassé à l'état fermé quand il aurait reçu le prochain bit sur le même fil. Ce genre de circuit est réalisé avec des circuits intégrés appelés portes logiques ou encore *latch*. Un verrou (*latch*) est un circuit électronique utilisé pour stocker de l'information dans les systèmes de logique séquentielle asynchrones. Un verrou peut stocker un bit de données. On trouve souvent plusieurs verrous dans un circuit dont certains ont des noms spéciaux, tels que le « quadruple verrou » (qui peut stocker quatre bits) et « le verrou octal » (huit bits). Les verrous sont des dispositifs qui n'ont aucune entrée d'horloge et changent l'état de leur rendement seulement en réponse à l'entrée de données, alors que les bascules (*flip-flops*) ont des entrées de données mais changent l'état de leur rendement seulement en réponse à une entrée d'horloge. Le plus simple des verrous logiquement est le verrou SR, où S vaut *SET* et R *RESET*. Le verrou est construit d'une paire de portes logiques

inter-connectées NI (négatif de OU). Le bit stocké est présent sur la sortie Q. Normalement, en mode de stockage, les entrées de S R sont toutes les deux en mode bas, et le retour (*feedback*) maintient les sorties Q q dans un état constant, q étant le complément du Q. Si S (*SET*) est en mode haut tandis que R est tenu bas, alors le rendement de Q est obligatoirement haut, et reste haut quand S redevient bas. D'autre part, si R est en mode haut tandis que S est tenu bas, alors Q est obligatoirement bas, et reste bas quand R redevient bas. Si S et R sont mis en mode haut en même temps, les deux conditionneurs NI donneront des zéros en sortie, menant à une oscillation afin que cette condition soit évitée. Voici le schéma type du système *latch* décrit ci-dessus :

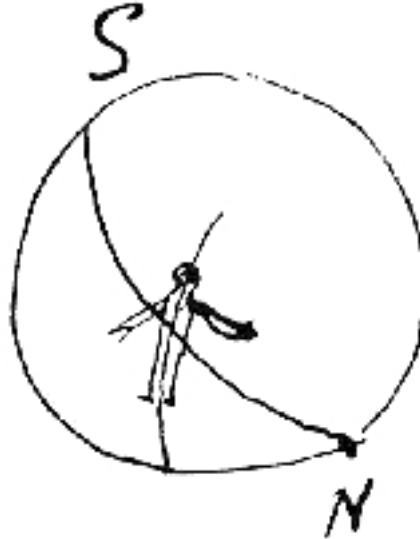


(SR-NOR-latch.jpg)

Dans le cas de notre circuit, la préoccupation est qu'on va alimenter un relais qui à une bobine sous 24V. La vitesse de commutation peut être non négligeable et nécessite un choix de composants rapides. Il y a deux types de relais : les relais mécaniques et les relais statiques. Un relais statique est un composant où il n'y a pas de pièce en mouvement. C'est un composant construit suivant la physique des semi-conducteurs. Ce type de composant est beaucoup plus cher, la différence de prix est considérable : un relais statique qui coupe un courant de 10 ampères sous 250V en entrée coûtera 115 euros alors qu'un relais mécanique équivalent coûtera moins d'un euro. Les relais statiques appartiennent à une technologie moderne et plus sophistiquée mais ils coupent généralement moins bien les charges inductives. En terme d'électronique, nous aurions plutôt eu tendance à utiliser des relais statiques car ils sont plus rapides, mais Monsieur BERARD ayant déjà acheté une grande quantité de relais mécaniques, nous étions donc amenés à utiliser ces derniers. En ce qui nous concerne, en sortie de la carte nous aurons des lampes halogènes basse tension (12V) nécessitant un transformateur. Ce transformateur est une charge inductive. Une charge inductive est une charge non directement voulue, issue des effets secondaires du passage du courant principal dans une bobine. Lors de ce passage un courant supplémentaire se crée, de vecteur orienté différemment. Ainsi le courant principal induit un courant de type différent affectant la ligne. Le problème des charges inductives est qu'elles sont relativement mal gérées par les relais statiques car le circuit est troublé par ce courant perpendiculaire supplémentaire, en désorganisant dans une certaine mesure « l'algorithmique » interne. D'autre part, le remplacement d'un relais doit pouvoir se faire aisément. Dans ce cas, le relais mécanique reste la meilleure solution, étant monté sur un support standard permettant de l'emboîter dans un *rack*.

### VI.3- Quels sont les impératifs d'un relais mécanique ?

Un relais mécanique se compose d'un certain nombre de contacts physiques mobiles qui sont pilotés par une petite bobine. Il y a un bobinage autour d'un noyau de fer doux. Quand on envoie du courant sur un des pôles de la bobine, le courant va parcourir le bobinage et va ressortir de l'autre côté. Au passage, comme il y a un nombre élevé de spires, il y a un effet induit par le courant qui crée un champ magnétique, c'est ce qu'on appelle la règle des trois doigts (le bonhomme d'Ampère).



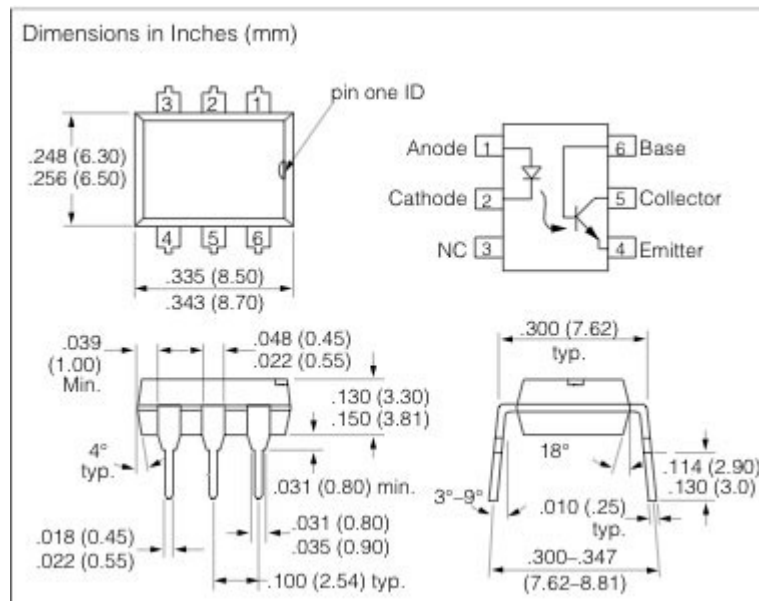
(bonhomme-ampere.jpg - Le "bonhomme": dessin d'Ampère - Archives de l'Académie des Sciences)

Ce champ magnétique fait bouger une petite barrette métallique qui sert d'interrupteur. Les contacts en cuivre sont traités avec une matière spéciale qui permet de ne pas créer d'étincelles (ou le moins possible). Cette bobine a des caractéristiques physiques en terme d'électronique : elle produit un appel de courant au démarrage et surtout, quand on relâche le relais, la tension subit comme une contre-réaction qui fait que si l'on alimente le relais en 24V, au moment où l'on va couper le courant, la tension va augmenter, pendant un temps qui est court, mais de façon sérieuse et cela peut détruire le circuit qui est en amont. Pour éviter cela, on intègre au circuit électronique ce qu'on appelle une "diode de roue libre". Ce dispositif récupère la tension et la renvoie dans la bobine, mais dans le sens contraire (c'est la sortie qui va renvoyer quelque chose, la sortie du relais va être renvoyée sur l'entrée). Bien entendu, en temps normal, si l'on connecte la sortie sur l'entrée, on crée un court-circuit. Pour éviter cela on met une diode entre la sortie et l'entrée du relais. Comme la diode est un composant qui ne conduit que dans un sens, le courant ne fera pas demi-tour. Une diode est un composant marqué par une bague, la bague marque le sens de coupure du courant. Dans une *led*, qui est également une diode électroluminescente, il y a une patte qui est plus grande que l'autre, et c'est par cette patte que le courant doit entrer pour sortir par la plus petite. Si l'on venait à alimenter la *led* du côté de la petite patte, aucun courant ne sortirait de l'autre côté. C'est un composant qui est très pratique car il peut sélectionner le sens du courant. Fort de cette caractéristique, l'idée est de rajouter une diode sur le relais afin que lorsque l'on coupe le courant, la

sortie n'en renvoie pas sur l'entrée. En ajoutant également une résistance on crée un amortissement encore plus rapide de ce courant perturbateur. Les relais achetés par Monsieur BERARD comportent déjà la diode de roue libre ainsi que la résistance. Notre relais est donc protégé à la coupure. Ce sont des relais OMERON.

Pour activer le contact du relais, nous avons besoin d'une tension proche de 24V. Ce contact va couper un circuit secondaire sous 220V qui va alimenter les circuits des lampes. Le problème est que nous avons 3.3V en sortie des broches. Cela signifie qu'il va falloir amplifier les 3.3V afin d'alimenter la bobine sous 24V. La deuxième contrainte est qu'il serait bon de protéger la carte GLUION de retours de courant qui pourraient se produire par des manipulations malencontreuses par exemple. Le courant sous 220V ne doit également surtout pas atteindre la carte GLUION sinon cette dernière serait totalement détruite !

Pour éviter cela on ajoute, en tampon intermédiaire, un optocoupleur. Un optocoupleur est un module enchâssant une surface photosensible alignée, sans contact, avec une petite *led*. C'est comme un transistor qui serait piloté par de la lumière. L'information provenant de la carte est ainsi isolée électriquement de la partie relais. La carte GLUION, en sortie des broches, va allumer la *led* qui va fournir de la lumière. A ce moment là, l'optocoupleur va laisser passer les 24V qui nous permettent d'alimenter notre relais et donc le déclencher. Lorsque la carte GLUION va arrêter de tenir cette tension de 3.3V, la *led* interne à l'optocoupleur s'éteindra et le courant 24V cessera. Cela aura pour effet immédiat de relâcher le relais et donc de couper l'alimentation des lumières. Notre optocoupleur est le modèle 4N37. Vous en trouverez une documentation au format .pdf (appelée aussi *data sheet*) dans le dossier DocEnPlus, sur le CD-Rom. En dehors des 2 broches pour alimenter la *led* interne à l'optocoupleur, nous avons aussi deux broches qui sont l'émetteur et le collecteur. En voici un schéma :

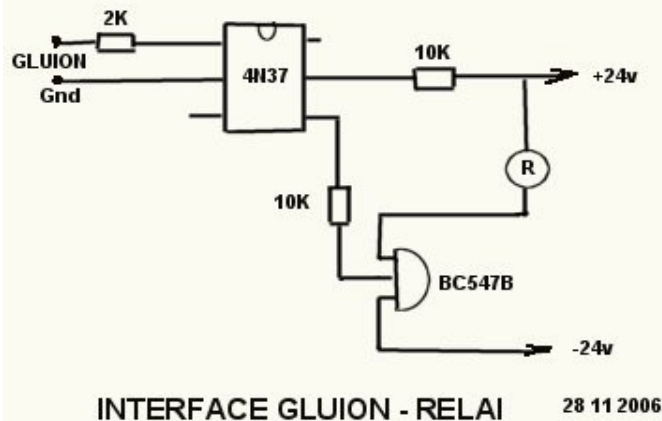


(schema4N37.jpg)



#### VI.4- Qu'est-ce qu'un transistor ?

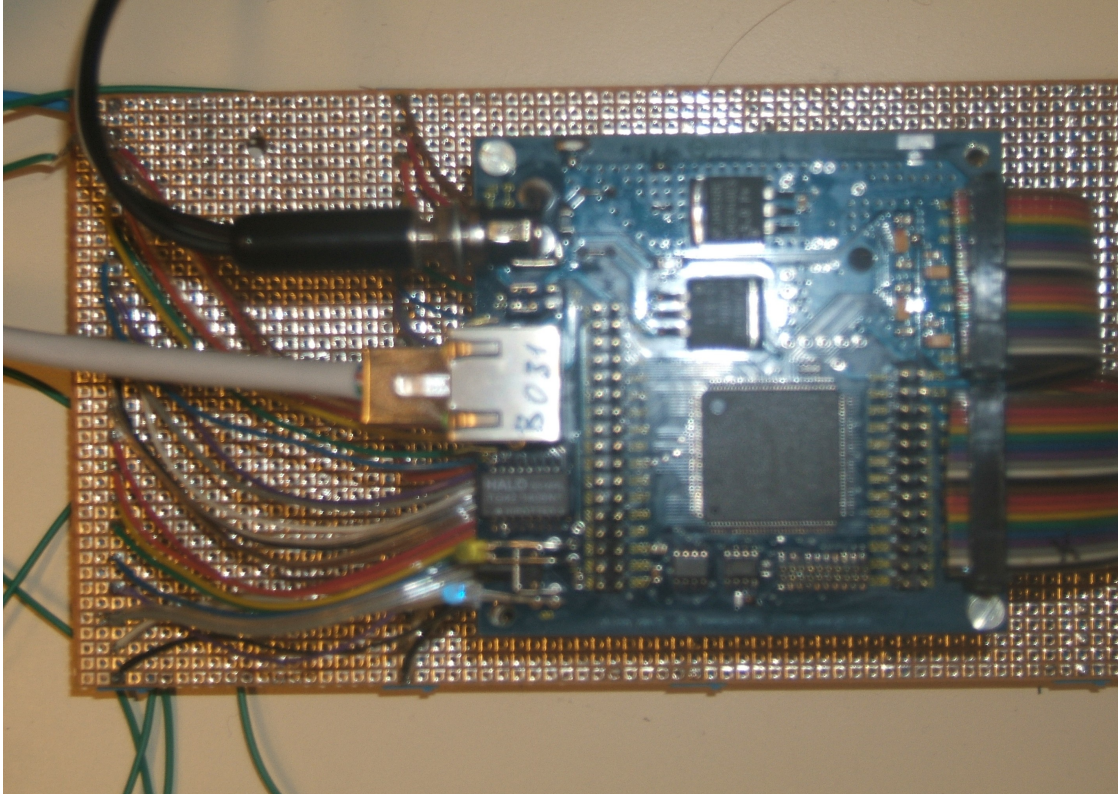
Si un optocoupleur est comme un transistor, alors qu'est-ce qu'un transistor ? Il comporte 3 broches qu'on appelle la base, le collecteur et l'émetteur. Un transistor permet beaucoup de montages électroniques différents. Suivant les composants qui y sont reliés, le transistor peut avoir diverses fonctions. La fonction qui nous intéresse est la fonction où le transistor est dit en mode saturé. Si l'on amène du courant à la base, le courant va pouvoir passer du collecteur à l'émetteur. Il fait un peu office de porte dans ce cas là. En mode saturation, si le courant de commande envoyé à la base est d'une certaine intensité, un autre courant va pouvoir aller du collecteur à l'émetteur. Si le courant de commande est en dessous de cette tension, il ne passera pas. C'est la condition qui nous intéresse : on alimente notre transistor par un courant de commande de manière à ce qu'un courant puisse passer du collecteur à l'émetteur, et ce courant ira alimenter la bobine du relais. Le courant de commande vient de la carte GLUION par l'intermédiaire de l'optocoupleur. Quand la lumière de l'optocoupleur va s'éteindre, l'optocoupleur n'alimentera plus la base du transistor, ce qui aura pour effet de couper le courant entre collecteur et émetteur, et donc de couper le relais. Un transistor fonctionne comme une *led* entre la base et l'émetteur, c'est à dire que le courant peut passer dans un sens mais pas dans l'autre. La tension de saturation est une valeur normalisée de 0.7V. A partir de 0.7V, le transistor est saturé. Dans notre montage, nous utilisons un transistor de référence BC547B qui peut recevoir une tension maximale entre collecteur et émetteur de 45V. Ce qui est très bien pour nous car nous voulons aller jusque 24V. Notre transistor est donc suffisant pour permettre de fournir les 24 Volts nécessaires à l'alimentation de la bobine de notre relais mécanique, et donc de permettre à notre circuit de fonctionner comme nous le souhaitons. Voici un schéma du circuit électronique tel qu'il est actuellement configuré :



(InterfaceGluionRelais.jpg)

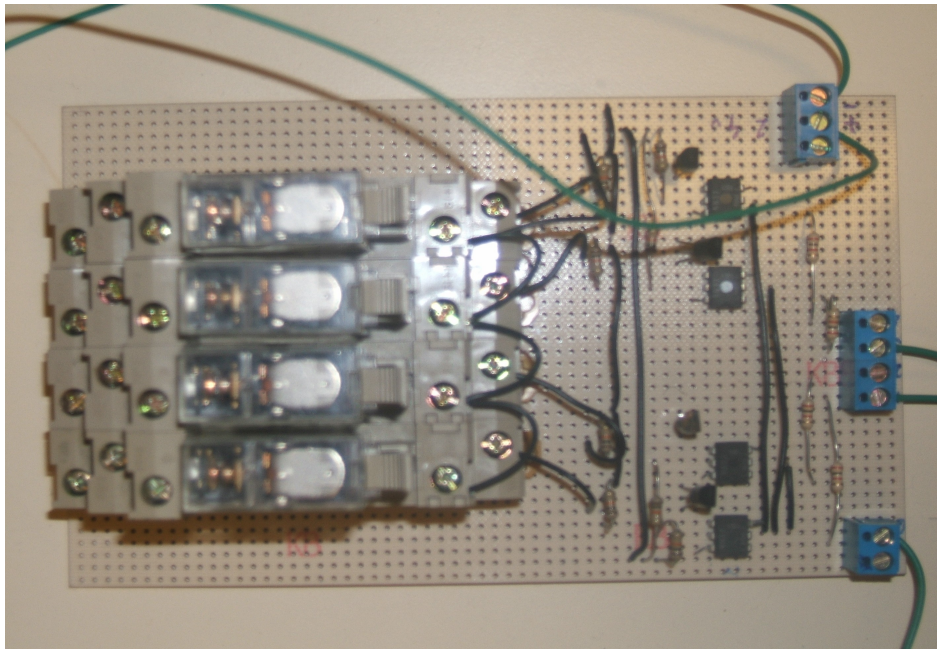
Ce schéma sera amené à changer car Monsieur KARTADINATA nous a donné des informations erronées, ce qui va certainement nous obliger à y apporter des modifications pour le contrôle des lumières en mode continu. Voici enfin quelques images qui illustrent les informations précédentes et montrent le circuit

électronique tel qu'il est actuellement :



(GluionConnected.JPG)

La carte GLUION est connectée par Ethernet à l'ordinateur et alimentée en 9V par une alimentation externe. Le gros composant carré est le FPGA. C'est lui qui gère la configuration des entrées/sorties que vous pouvez voir sur la droite et la gauche.



(circuitElectro.JPG)

Cette illustration représente le circuit électronique tel qu'il est actuellement. Les fils connectés en haut à droite proviennent de l'alimentation 24V destinée aux relais, au milieu à droite ce sont les 3.3V de la carte qui contrôlent les optocoupleurs et en bas à droite, c'est la masse. Sur la gauche se trouvent quatre

contacts. Quant aux quatre petits rectangles noirs, il s'agit des optocoupleurs

Voici un exemple d'alimentation électrique utilisée. Il s'agit ici de celle fournissant le 24V au montage électronique :



(Alim24V.JPG)

Enfin, dans la partie suivante, je vais expliquer quelques points de base au sujet des protocoles de communication comme par exemple Ethernet et UDP.

## **VI.5- Protocoles de communication**

### **VI.5.1- Définitions du mot protocole selon le Petit Robert :**

- 1- Recueil de formules en usage pour les actes publics, la correspondance officielle.
- 2- Recueil des règles à observer en matière d'étiquette, préséance, dans les cérémonies et les relations officielles.

### **VI.5.2- Définition d'un protocole de communication selon Wikipedia, l'encyclopédie libre sur internet**

« Dans les réseaux informatiques et les télécommunications, un protocole de communication est une spécification de plusieurs règles pour un type de communication particulier. Initialement, on nommait protocole ce qui est utilisé pour communiquer sur une même couche d'abstraction entre deux machines différentes. Par extension de langage, on utilise parfois ce mot aussi aujourd'hui pour désigner les règles de communication entre deux couches sur une même machine. Les protocoles de communication les plus utilisés sont les protocoles réseau ». Exemples :

IP : *Internet Protocol*

TCP : *Transport Control Protocol*

LDAP : *Lightweight Directory Access Protocol* (accès à annuaires)



### VI.5.3- Le concept

Communiquer ne consiste pas seulement à transmettre des informations (sans quoi les protocoles seraient inutiles). Il faut s'assurer que :

- 1. l'interlocuteur sache que vous avez quelque chose à transmettre (par exemple, au téléphone : vous composez son numéro pour faire sonner son combiné)
- 2. qu'il soit prêt pour cela (téléphone : vous attendez qu'il décroche)
- 3. qu'il situe votre communication dans son contexte ("Je t'appelle pour la raison suivante...")
- 4. qu'un éventuel destinataire final y soit identifié ("Peux-tu prévenir Michel que...")
- 5. que le correspondant s'assure d'avoir bien compris le message ("Peux-tu me répéter le nom ?")
- 6. que des procédures d'anomalies soient mises en place ("Je te rappelle si je n'arrive pas à le joindre")
- 7. qu'on se mette d'accord sur ce qu'est la fin de la communication ("Merci de m'avoir prévenu").

Cette méta-communication n'est autre que la mise en œuvre d'un protocole.

### VI.5.4- Qu'est-ce que Ethernet ?

C'est au départ une technologie de réseau local permettant que toutes les machines d'un réseau soient connectées à une même ligne de communication, formée de câbles cylindriques (câble coaxial ou paires torsadées). Le standard qui a été le plus utilisé dans les années 1990 et qui l'est toujours est le 802.3 de l'*Institute of Electrical and Electronics Engineers* (IEEE) (maintenant aussi adopté comme norme internationale ISO/CIE 8802-3). Le nom Ethernet vient de éther, milieu mythique dans lequel baigne l'univers, et *net*, qui est une abréviation de *network* (réseau en anglais).

Ethernet est un protocole de réseau informatique à commutation de paquets. Sur les réseaux numériques, les protocoles gèrent le codage, l'assemblage, le transport, le décodage et le ré-assemblage de l'information véhiculée par commutation de paquets discontinus d'informations, en général de tailles fixes. Ethernet fait partie d'une des cinq couches qui constituent la pile *Transmission Control Protocol/Internet Protocol* (TCP/IP).

Le schéma suivant situe divers protocoles de la pile TCP/IP :

5 Application	ex.	HTTP, FTP, DNS, SMPT, Telnet
4 Transport	ex.	TCP, UDP, RTP, ATP (Apple Talk Protocol)
3 Réseau		pour TCP/IP il s'agit de IP (les protocoles requis comme ICMP et IGMP fonctionnent au-dessus d'IP, mais peuvent quand même être considérés comme faisant partie de la couche réseau ; ARP ne fonctionne pas au-dessus d'IP)
2 Liaison	ex.	Ethernet, Token Ring, RNIS, WI-FI etc.
1 Physique	ex.	réseau physique, et techniques de codage, son, vidéo...

La couche «liaison de données» gère les communications entre deux machines adjacentes, i.e. directement reliées entre elles par un support physique. Elle est chargée du regroupement de *bits* isolés en trames ou de la délimitation de ces trames dans un flot continu de *bits*. Les en-têtes des trames Ethernet, par exemple, contiennent des champs qui indiquent à quelle(s) machine(s) du réseau un paquet est destiné. La liaison de données est aussi souvent chargée de la détection des erreurs de transmission et parfois de leur correction.

#### VI.5.5- Revenons au patch

Si vous vous souvenez, dans les premières pages de ce mémoire, je disais qu'un technicien MOELLER était venu au CNSMDL avec ses nouveaux modèles de contrôleurs pilotables via Ethernet. Nous avons alors essayé de communiquer avec les contrôleurs en utilisant l'objet **OpenSoundControl** (OSC). Le problème que nous avons dû résoudre était que le contrôleur attendait des informations au format UDP alors que nous lui envoyions nos informations sous un autre format (format propre à OSC).

OSC est en fait un certain format d'encapsulation de données. Le format OSC peut être envoyé par UDP ou TCP/IP à l'aide de l'objet **otudp write** et il devra être reformaté à la sortie à l'aide de l'objet **otudp read**. En gros, nous avons passé l'étape 1 ci-dessus et nous étions bloqués à l'étape 2.

**OpenSoundControl** est un objet très fiable, là n'est pas le problème. Il fut même pendant longtemps la référence pour transmettre des messages Max sur un réseau Ethernet mais il n'était pas l'objet le plus approprié pour la simplicité des messages que nous voulions envoyer et la lourdeur du système à mettre en place pour utiliser OSC. De plus, l'utilisation du message `resetallthewaymode` en rend l'utilisation risquée, ce qui n'est pas ce que nous cherchons.

Il existe d'autres objets qui permettent de communiquer sur un réseau Ethernet comme **otudpsend** et **otudpreceive** ou encore **udpsend** et **udpreceive**. Notre application utilise les objets **udpsend** et **udpreceive**. A la base, j'ai choisi ces objets pour leur simplicité d'utilisation ainsi que pour leur stabilité. L'objet **udpsend** ne prend que deux arguments :  
adresse IP pour l'envoi des données – numéro de port Ethernet.

L'objet **udpreceive** quant à lui n'en prend qu'un seul qui est le numéro de port Ethernet sur lequel les informations **udpsend** doivent lui parvenir. Rien de plus simple et la communication est immédiate à partir du moment où les ordinateurs sont reliés par un câble Ethernet croisé bien entendu (à moins que d'utiliser un *hub*).

Depuis la version 4.6 de Max, ce choix d'utiliser **udpsend** et **udpreceive** s'est avéré le meilleur possible. Les objets **udpsend** et **udpreceive** sont distribués avec l'application ce qui évite d'avoir à les joindre au dossier et d'en suivre le développement pour d'éventuels portages. Même le CNMAT qui est pourtant l'inventeur du protocole OSC sur UDP écrit sur son site *web* (<http://www.cnmatic.berkeley.edu/OpenSoundControl/Max/>) :

« *The most reliable implementation of the UDP part is the `udpsend` and `udpreceive` externals that are now distributed with Max 4.6. CNMAT recommends these over our own (or anybody else's) older implementations. Why?*

- *1. because they interact cleanly with Max's threading and scheduling systems*
- *2. because they were written recently rather than being a port of a port...*
- *3. because they're cross-platform. »*

Eh oui, c'est bien ça, ils sont également multi-plateformes ce qui veut dire qu'ils fonctionnent aussi bien sous Windows XP que sous Mac OS X. Jusque là nous étions obligés d'avoir une version Mac et une version PC de notre application simplement à cause de problèmes de compatibilité entre les deux systèmes d'exploitation. Cela tombe on ne peut mieux.

A titre indicatif, je vous invite à vous reporter aux photographies annexées, où vous trouverez une copie d'écran du patcher que j'utilisais encore récemment avec OSC (photo OSC\_basics).

#### VI.5.6- Qu'est-ce qu'UDP ?

UDP signifie *User Datagram Protocol* (protocole de datagramme utilisateur). Si vous reprenez le tableau de la page précédente sur les divers protocoles de la pile TCP/IP, vous trouverez UDP dans la couche transport. Les protocoles de la couche de transport peuvent résoudre des problèmes comme la fiabilité des échanges (« est-ce que toutes les données sont arrivées à destination ? ») et assurer que les données sont rangées dans l'ordre chronologique d'émission. Dans la suite de protocoles TCP/IP, les protocoles de transport déterminent aussi à quelle application chaque paquet de données doit être délivré.

Le rôle de ce protocole est de permettre la transmission de paquets (aussi appelés datagrammes) de manière très simple entre deux entités, chacune étant définie par une adresse IP et un numéro de port (pour différencier divers utilisateurs ou différentes applications sur la même machine). UDP est généralement utilisé par des applications de diffusion multimédia (audio et vidéo, etc). Typiquement, il sert à transporter des vidéos pour que l'on puisse synchroniser la lecture des images et du son directement, sans les stocker préalablement. Son plus gros atout : la rapidité.

#### VI.5.7- Qu'est-ce que le DMX ?

Le DMX (*Digital MultipleXing*) est un protocole de multiplexage de données. Il est essentiellement utilisé pour le contrôle de l'éclairage dynamique d'événements, en complément de la sonorisation (concerts, plateaux télé, spectacle son et lumière).

Avant le DMX, la commande d'éclairage en théâtre se faisait généralement par la commande entre 0 et 10 V de gradateurs reliés à un ou plusieurs projecteurs branchés en parallèle : 0V pouvant signifier l'extinction totale, 10V l'allumage complet. Certains gradateurs sont équipés de façon à fournir des tensions comprises entre 0 et 10V (pour des effets d'ambiance en n'allumant le

projecteur qu'à moitié par exemple). C'est en 1986 que l'*Engineering Commision* de l'*United States Institute for Theatre Technologies* a mis en place une nouvelle norme numérique, appelée DMX.

Le protocole DMX512 (norme RS 485) permet de contrôler 512 canaux en affectant à chacun une valeur comprise entre 0 et 255. La transmission se fait de façon sérielle, et chaque appareil reçoit l'ensemble des 512 valeurs (ce qu'on appelle une "trame" DMX) et renvoie cette trame à l'appareil suivant, ne s'occupant que des informations qui lui sont destinées (comme dans le protocole MIDI).

La transmission se fait par un câble XLR de type 5 (standard de la norme) ou 3 broches. La fréquence de rafraîchissement est de 44 Hz (ce qui signifie que la trame est envoyée 44 fois par seconde). Le signal est électriquement en 5 volts.

Pour plus d'informations sur le DMX, voici un lien vers un site internet canadien relativement clair et complet :  
<http://www.jonathan-morin.qc.ca/site/normedmx.html>

La norme DMX est cependant déjà limitée à cause de sa sensibilité électrique, de son câblage fastidieux, et du nombre limité de canaux que l'on peut commander. Une nouvelle norme a été créée, nommée Artnet, qui intègre les trames DMX dans des paquets Ethernet, et permet donc un câblage beaucoup plus facile comme pour un réseau informatique ainsi que l'utilisation d'équipements avancés comme par exemple le WI-FI...

## **VII- Version actuelle**

Ce chapitre est certainement le plus important. C'est le chapitre dans lequel figure la description détaillée de chacun des patchers qui constituent le patch global MMS (*MultiMedia Sequencer*). Dans ce chapitre, nous allons étudier les points suivants :

- comment installer notre application MMS ?
- comment paramétrer Max-MSP pour pouvoir utiliser notre séquenceur multimédia ?
- comment paramétrer votre ordinateur sous Mac OS X et/ou Windows XP ?
- comment MMS fonctionne-t-il et comment s'en servir ?

J'insiste sur le mot « actuelle » du titre de ce chapitre car il est bien entendu que la version que je vais décrire ici aura déjà certainement subi quelques modifications plus ou moins importantes au moment où vous lirez ce mémoire. Il m'est honnêtement difficile de faire autrement car un logiciel a de nombreux points communs avec une partition de musique : l'un comme l'autre pourront toujours être revus, corrigés et améliorés et cela pratiquement sans limite de temps. De plus, chaque nouvelle version de Max-MSP apporte son lot de nouvelles fonctions. Ces nouvelles fonctions me permettent bien souvent de rendre l'interface utilisateur du logiciel encore plus intuitive et/ou simple d'utilisation. Il serait dommage de ne pas en faire profiter notre logiciel. Enfin, alors que j'écris ce mémoire, je continue à me former de diverses façons à la programmation informatique appliquée à la musique. Chaque chose que j'apprends aujourd'hui peut être utile à ce mémoire et à l'amélioration de notre séquenceur multimédia. Je ferai de mon mieux pour tenir ce mémoire à jour afin que la version que vous lirez soit la plus récente possible.

Encore une fois, vous n'êtes pas obligé d'installer ce logiciel pour pouvoir suivre le déroulement du mémoire. Cependant, si vous voulez l'installer, veuillez suivre les quelques étapes suivantes.

### **VII.1- Installation de notre séquenceur multimédia**

- insérez dans votre ordinateur le CD-Rom joint à ce mémoire si cela n'est pas déjà fait et ouvrez le dossier MMS. Vous trouverez dans ce dossier plusieurs sous-dossiers ainsi que des fichiers. Veuillez, je vous prie, commencer par lire le fichier nommé LICENCE\_GPL
- une fois que vous avez pris connaissance de chacun des points de la licence pré-citée, copiez le fichier de nom MMS\_New\_Session.pat à l'adresse suivante :  
/Applications/MaxMSP 4.6/patches/templates  
(ceci est un chemin d'accès Macintosh mais la seule différence sur PC devrait être que chaque subdivision de l'arborescence est représenté par \ plutôt que /)
- copiez maintenant le dossier MMS\_All sur votre disque dur à un emplacement où il vous sera commode de situer vos montages (ex. : un dossier où se trouvent tous vos projets Max-MSP)



comme vous voyez, le dossier MMS\_All contient lui-même plusieurs dossiers et fichiers.

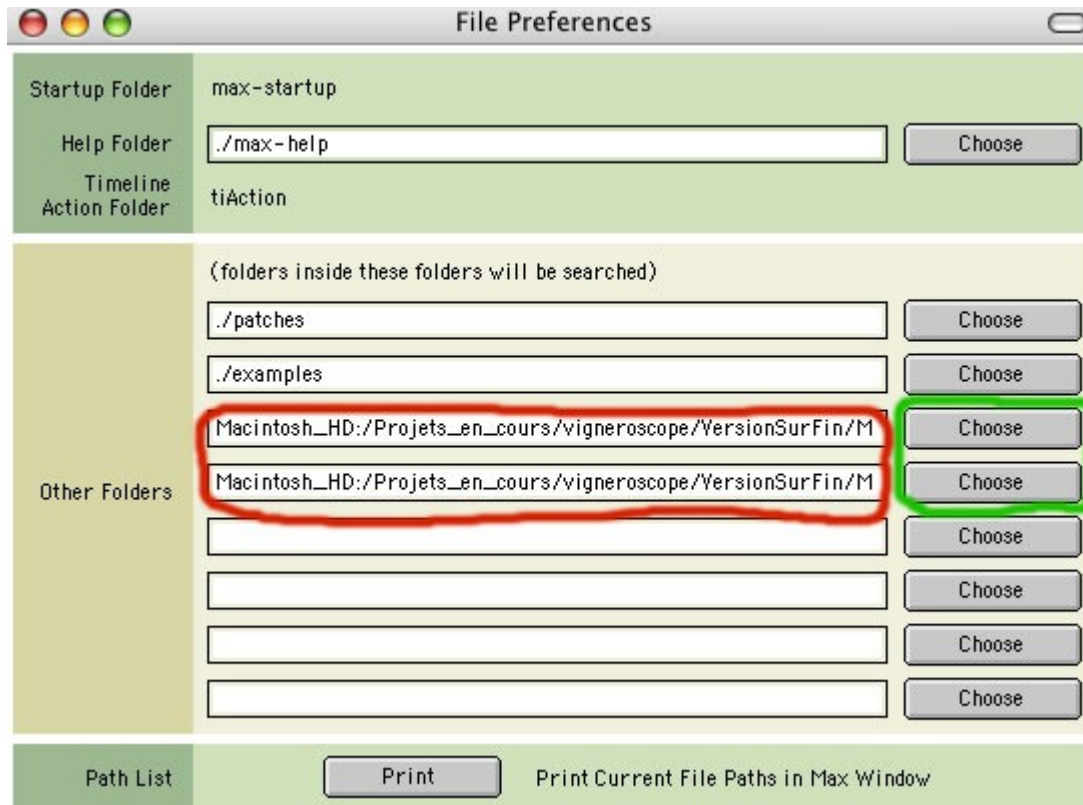
- \*\*\* MMS\_lib contient des fichiers auxquels notre séquenceur fera continuellement référence, on appelle cela des bibliothèques. Je reviendrai sur pratiquement chacune d'entre elles plus tard
- \*\*\* My\_Sessions est le dossier dans lequel je vous invite à mettre tous vos différents montages, vos différentes sessions pour utiliser un terme propre aux séquenceurs
- \*\*\* Soundfiles est le dossier dans lequel je vous invite à placer tous les fichiers audio que vous utiliserez dans vos différents montages afin que tout ce qui concerne vos sessions MMS soit rassemblé en un seul et même endroit sur le disque de votre ordinateur
- \*\*\* Video est le dossier dans lequel vous devrez placer chacun des fichiers vidéo que vous utiliserez dans vos différents montages afin que Max sache où les trouver lors de la lecture de vos montages. (Video sans accent car un bon chemin d'accès sur un système d'exploitation ne devrait jamais contenir le moindre caractère spécial, c'est à dire de caractère dont le code ASCII ne soit pas compris entre 0 et 255)

Voilà pour ce qui concerne l'installation de MMS.

## VII.2- Configurer Max-MSP pour l'utilisation de MMS

Vos dossiers d'installation sont enregistrés sur votre disque dur. Nous allons maintenant dire à l'environnement Max-MSP où il va trouver ces dossiers et fichiers sur le disque dur.

Démarrez l'environnement Max-MSP et déplacez-vous dans le menu Options afin d'y trouver *File preferences*. Lorsque vous cliquez sur *File preferences*, une fenêtre comme ci-dessous apparaît sur votre écran :



(FilePrefMax.jpg)

La copie d'écran ci-dessus est telle qu'elle est configurée sur mon propre ordinateur. Les trois premiers champs sont donnés par défaut par Max-MSP, ils correspondent aux chemins d'accès dont Max-MSP a besoin pour pouvoir fonctionner correctement, il ne faut donc surtout pas les modifier. Pour ajouter un chemin, utilisez les boutons *Choose* que j'ai entourés, à droite, sur la copie d'écran. Encore une fois, n'effacez aucun chemin existant mais ajoutez vos nouveaux chemins là où les champs sont encore vides.

Pour notre séquenceur MMS, nous aurons besoin d'ajouter deux chemins pour l'instant :

- un premier qui pointe sur la bibliothèque MMS\_lib qui se trouve dans le dossier MMS\_All. MMS\_lib est la bibliothèque à laquelle MMS se réfère constamment au démarrage de MMS ainsi qu'à chaque montage créé
- un second pointant sur le dossier Video qui se trouve dans le dossier MMS\_All et qui permettra à Max de trouver les fichiers de type vidéo chaque fois que MMS trouvera un plug-in vidéo dans un montage

Une fois vos chemins correctement entrés, fermez cette fenêtre *File preferences* et quittez l'environnement Max-MSP.

Quitter l'environnement Max-MSP permet à Max de prendre en considération les changements que vous venez d'apporter en ajoutant vos chemins d'accès personnels. En effet, Max tiendra compte des nouveaux chemins d'accès seulement après avoir été redémarré. Vous pouvez maintenant relancer l'environnement Max-MSP.

Cliquez sur le menu *File* de Max-MSP et positionnez votre pointeur de souris sur *New*. Un second menu apparaît alors dans lequel devrait apparaître notre séquenceur multimédia sous le nom : *MMS\_New\_Session*. Sélectionnez *MMS\_New\_Session* dans ce même menu et Max-MSP ouvre aussitôt une session vierge MMS.

### **VII.3- Comment paramétrer votre système d'exploitation pour l'utilisation de MMS**

#### **VII.3.1- Préférences système pour Mac OS X** (version 10.3.9 minimum recommandée et 10.4.n conseillée)

Comme nous l'avons vu au chapitre cinq, MMS utilise le réseau Ethernet pour communiquer avec la carte externe GLUION. C'est cette carte qui va recevoir les messages de contrôle venant de MMS et va les transmettre au circuit électronique qui permet l'alimentation électrique du réseau lumière, que ce soit en mode « contrôle tout ou rien » ou en mode « contrôle continu ».

Afin de pouvoir utiliser le réseau Ethernet avec MMS, il va falloir configurer Mac OS X de manière à ce que l'ordinateur Macintosh et la carte GLUION puissent communiquer. Pour ce faire, suivez les indications suivantes :

- ouvrez les Préférences Système que vous pouvez trouver soit dans votre *Dock* soit dans le menu *Finder* et en cliquant sur Préférences
- cliquez sur Réseau
- dans le menu déroulant Configuration de la fenêtre réseau, sélectionnez « Nouvelle configuration ». Nommez-la par exemple : GLUION
- dans le menu déroulant Afficher de la fenêtre réseau, sélectionnez « Ethernet intégré »
- dans le menu TCP/IP, dans Configurer IPv4, sélectionnez « Manuellement ». Le champ Adresse IP doit être configuré comme suit : 192.168.5.80 et le champ Sous-réseau doit être porté à 255.255.255.0, les autres champs sont vides. Quant au champ Adresse IPv6, il doit être configuré sur « Automatiquement »
- PPPoE est vierge
- Apple Talk n'est pas activé
- Proxy est vierge. Seul « Utiliser le mode FTP passif (PASV) » est coché
- Ethernet est configuré comme « Automatiquement »
- cliquez maintenant sur « Appliquer » en bas de la fenêtre
- vous pouvez quitter les Préférences Système

Reliez la carte GLUION à l'ordinateur grâce à votre câble Ethernet. Afin de vérifier que votre Macintosh et la carte GLUION communiquent bien, vous pouvez ouvrir un terminal Unix que vous trouverez dans : Applications/Utilities. Attention, la carte et l'ordinateur doivent absolument être reliés avant de faire ce qui suit !!!

Lancez l'application Terminal et tapez la commande Unix suivante :

```
arp -a (suivi de entrée sur votre clavier)  
La carte GLUION devrait apparaître comme suit :  
192.168.5.77    00.12.34.56.78.90
```

Pour plus d'informations sur la commande Unix arp, vous pouvez taper dans votre terminal Unix la commande suivante :  
man arp

Quittez votre terminal en exécutant la commande suivante :  
exit

### **VII.3.2- Configuration réseau pour Windows XP**

Comme nous l'avons vu au chapitre cinq, MMS utilise le réseau Ethernet pour communiquer avec la carte externe GLUION. C'est cette carte qui va recevoir les messages de contrôle venant de MMS et va les transmettre au circuit électronique qui permet l'alimentation électrique du réseau lumière, que ce soit en mode « contrôle tout ou rien » ou en mode « contrôle continu ».

Afin de pouvoir utiliser le réseau Ethernet avec MMS, il va falloir configurer Windows XP de manière à ce que votre ordinateur et la carte GLUION puissent communiquer. Pour ce faire, suivez les indications suivantes :

- en bas de votre écran, vous avez votre menu Démarrer. Cliquez dessus et choisissez « Panneau de configuration »
- allez dans « Connexions réseau et Internet »
- double-cliquez sur « Connexion au réseau local »
- une fenêtre apparaît, c'est la fenêtre des propriétés de connexion au réseau local. Cette fenêtre comporte deux onglets, celui qui nous intéresse est l'onglet « Général »
- dans cet onglet « Général », vous avez un menu déroulant, faites-en descendre l'ascenseur jusqu'à ce que vous trouviez « Protocole internet (TCP/IP) »
- sélectionnez « Protocole internet (TCP/IP) » puis cliquez sur « Propriétés » juste en-dessous
- une seconde fenêtre apparaît, c'est la fenêtre « Propriétés de protocole internet (TCP/IP) »
- dans l'onglet « Général » sélectionnez « Utiliser l'adresse IP suivante » et entrez l'adresse 192.168.5.80. Le Masque de sous-réseau doit être le suivant : 255.255.255.0 (Windows le donne par défaut habituellement)
- toujours dans cette fenêtre « Propriétés de protocole internet (TCP/IP) », cliquez sur « Avancé »
- vous aurez quatre onglets disponibles cette fois-ci (Paramètres IP, DNS, WINS, Options)

- cliquez sur WINS et dans « Paramètre NetBIOS », cliquez sur « Désactiver NetBIOS avec TCP/IP »
- appliquez en faisant OK
- appliquez encore une fois en faisant OK
- cliquez sur « Fermer »

Votre système d'exploitation Windows XP est maintenant prêt à fonctionner avec MMS.

Reliez la carte GLUION à l'ordinateur grâce à votre câble Ethernet. Afin de vérifier que votre PC et la carte GLUION communiquent bien, vous pouvez ouvrir un terminal que vous trouverez dans le menu Démarrer -> Exécuter. Attention, la carte et l'ordinateur doivent absolument être reliés avant de faire ce qui suit !!!

- lancez l'application Terminal. Une petite fenêtre apparaît dans laquelle vous devriez avoir un chemin d'accès sur-ligné
- effacez ce chemin d'accès et remplacez le par CMD
- faites OK. Une fenêtre terminal apparaît alors
- inscrivez-y la commande suivante :  
ipconfig
- tapez sur entrée de votre clavier

La carte GLUION devrait apparaître comme suit :

```
192.168.5.77    00.12.34.56.78.90
```

Quittez votre terminal en exécutant la commande suivante :  
exit

#### **VII.4- Comment l'application MMS fonctionne t-elle en interne ?**

Durant tous les sous-chapitres VII.4.N suivants, je vais décrire le fonctionnement interne de chacun des patchers qui constituent l'application MMS. C'est, si l'on veut, la partie théorique de l'explication de MMS. Le chapitre VIII, quant à lui, explique beaucoup plus simplement la mise en pratique de MMS et comment en utiliser l'interface graphique afin de pouvoir réaliser vos propres montages de contrôle multimédia.

##### **VII.4.1- Pourquoi un *template* ?**

Tout d'abord, pourquoi, lors de l'installation de MMS, vous ai-je fait installer MMS\_New\_Session.pat dans le dossier Templates de Max-MSP 4.6 ?

Les *Templates* en Max-MSP sont des *patchers* qu'on veut souvent utiliser comme point de départ d'autres travaux. Quand on choisit un *Template* à partir du menu *New* de Max-MSP, le patcher s'ouvre en mode édition. La fenêtre n'est pas nommée et n'est pas marquée comme modifiée de sorte que vous puissiez directement commencer à travailler dans ce même patcher et le sauvegarder où vous le désirez sans jamais effacer ou modifier le patcher source.

Jusqu'à présent, quand l'utilisateur de MMS voulait créer une nouvelle session de travail, il devait ouvrir une session vierge de référence, en changer le numéro et la sauvegarder sous un nom qui lui soit propre mais en faisant très attention de faire « enregistrer sous » et non « enregistrer » car sinon, il remplaçait la session vierge de référence par celle qu'il venait tout juste de créer. C'était particulièrement risqué car tout le monde est habitué à sauvegarder son travail en faisant « enregistrer » et non « enregistrer sous ».

J'ai décidé d'utiliser cette nouvelle fonction de Max-MSP il n'y a que peu de temps en réalité. Je cherchais un moyen de faire en sorte que l'utilisateur ne risque jamais d'effacer le *patch* source en sauvegardant son travail et je savais que c'était quelque chose qui pouvait très facilement arriver. Je voulais être sûr que quoi que fasse l'utilisateur, rien de dangereux ou d'important ne pouvait arriver malencontreusement au programme source (à la session vierge).

Le problème avec les *Templates* c'est, comme je le disais plus haut, que le patcher qui est ouvert en tant que *Template* est ouvert en mode édition et que je ne peux pas demander à l'utilisateur de fermer le mode édition du patcher avant de commencer à l'utiliser. Ce n'est pas pratique pour l'utilisateur, c'est risqué car une mauvaise manipulation est vite arrivée et qu'un patcher en mode édition est vulnérable. Enfin, ce n'est pas ce qui se fait de mieux en termes d'interface utilisateur.

Le gros avantage d'utiliser la fonction *Template* de Max-MSP c'est qu'une fois le *Template* ouvert, l'utilisateur n'a plus qu'à apporter les modifications qu'il désire puis sauvegarder sa nouvelle session en faisant « enregistrer » comme dans n'importe quel programme informatique. Le fait que la fenêtre du patcher *Template* ne soit pas marquée comme modifiée lors de l'ouverture permet de faire en sorte que le programme source, la session vierge, soit protégé de tout écrasement lors de la sauvegarde.

#### VII.4.2- Fermer le mode édition d'un *Template* Max-MSP

En temps normal, pour fermer le mode édition d'un patcher, plusieurs solutions sont offertes :

- le raccourci clavier pomme-e sur Macintosh ou ctrl-e sur PC
- edit dans le menu *View* de Max-MSP
- enfin cliquer sur le bouton prévu à cet effet par Max-MSP en haut de chaque fenêtre de patcher

Aussi pour communiquer avec la fenêtre de patcher, on utilise généralement l'objet Max **thispatcher** mais celui-ci n'a aucun message documenté qui permettrait d'ouvrir ou de fermer le mode édition d'un patcher. J'ai pourtant longtemps cherché au-delà des documentations car j'ai fait plusieurs fois l'expérience de découvrir des messages non répertoriés plus ou moins par hasard. J'en ai effectivement trouvé un pour l'objet **thispatcher**, c'est le message `edit`. si l'on envoie le message `edit` à l'objet **thispatcher**, celui-ci envoie le message suivant à la fenêtre Max :

error : patcher : missing arguments for message edit  
(arguments manquants pour le message edit).

Si l'objet **thispatcher** ne comprenait pas le message edit, cela apparaîtrait clairement dans la fenêtre Max sous forme d'un message d'erreur du genre :

error : thispatcher doesn't understand message edit.

Or ce n'est pas le cas, il attend bien des arguments que je n'ai toujours pas trouvés au moment où j'écris cette phrase.

Quand aucun objet ni aucune combinaison d'objets ne conviennent à ce qu'on essaie de faire, la seule solution qui reste, c'est de programmer soi-même un objet qui exécutera la tâche recherchée. Je le redis, c'est aussi ce qui fait la grande puissance de l'environnement de programmation Max-MSP. J'ai donc programmé mon propre objet en javascript à l'aide d'une fonction que m'a bien gentiment soufflée Emmanuel JOURDAN de l'IRCAM. Cette fonction est la fonction `this.patcher.locked`. Voici l'intégralité du code de mon programme javascript `thispatchlock` qui est interprété dans Max à l'aide de l'objet **js** :

```
// thispatchlock is a personal version of the ej.lock object by Emmanuel
// Jourdan.
// Emmanuel kindly gave me the basic clue on how to do this and i experimented
// until i got to the solution.
// This modified version of ej.lock is not better than the version by E.Jourdan.
// It's only my version.
// I can only recommend the use of Emmanuel Jourdan's object ej.lock
// Thanks Emmanuel for your help.
// thispatchlock locks or unlocks a patcher depending on the number it receives
// in its inlet.
// 1= locked 0 = unlocked

inlets = 1;
var a ;

function msg_int (a)
{
    this.patcher.locked = a ;
}
```

Vous pouvez trouver le code source de ce petit programme javascript soit sur le CD-Rom si vous n'avez pas installé MMS soit dans le dossier MMS\_lib qui se trouve dans le dossier MMS\_All. Le fichier en question se nomme `thispatchlock.js`. Bien entendu, si votre intention n'est pas d'apporter une modification positive au code source de `thispatchlock.js`, merci de ne rien changer à ce code.

Ce que fait ce programme est indiqué en en-tête : il y a une variable `a` qui peut être portée soit à 0 soit à 1. Si `a=1`, alors fermer le mode édition du patcher. Si `a=0`, ouvrir le mode édition du patcher dans lequel se trouve l'objet **js**.

### VII.4.3- Description de la fenêtre principale MMS

Quand vous ouvrez le patcher MMS\_New\_Session.pat qui se trouve dans le dossier *Templates* de Max-MSP 4.6 si vous avez installé MMS, une série de patchers s'ouvre. Tous ces patchers sont eux-mêmes des sous-patchers de la fenêtre où vous pouvez lire *Only Window To Save*. Voici une copie d'écran de cette fenêtre. Vous n'avez pas besoin de faire vous même ce qui suit. Il vous suffit de suivre à l'aide des copies d'écran qui illustrent les explications.

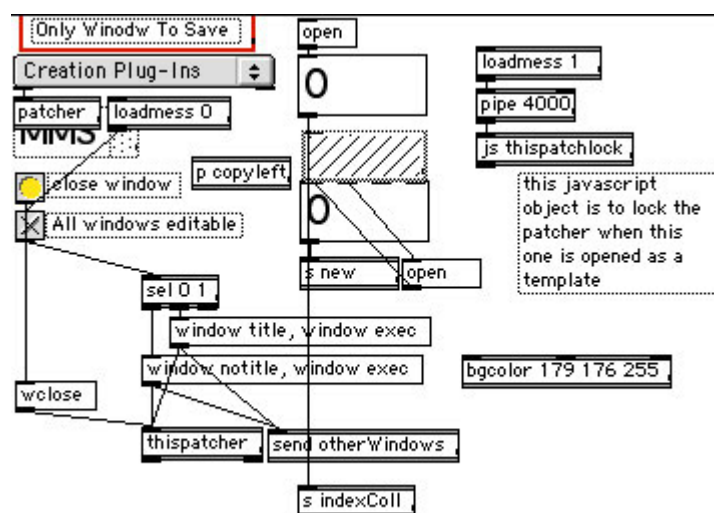


(MMS\_New\_Session\_Closed.jpg)

J'ouvre cette fenêtre de manière à ce qu'elle soit en mode édition et que vous puissiez voir tous les objets et connexions cachés. Pour cela, je fais comme suit :

- en bas à gauche du patcher MMS\_New\_Session, je clique sur le petit carré où vous pouvez lire *All windows editable*
- dans le menu *View* de Max-MSP, je clique sur *Edit*
- j'agrandis la fenêtre à l'aide des boutons en haut de fenêtre comme je le ferais avec n'importe quel traitement de texte par exemple

Le patcher MMS\_New\_Session occupe maintenant tout mon écran d'ordinateur et voici ce que je peux voir :



(MMS\_New\_Session\_Opened.jpg)

En haut à droite, vous pouvez voir l'objet **js** qui utilise notre petit programme javascript *thispatchlock.js* présenté à la section précédente. L'argument de l'objet **js** est le nom du fichier



javascript que **js** doit charger et exécuter. C'est pourquoi son argument est **thispatchlock**. Vous pouvez voir au passage que notre programme **thispatchlock.js** reçoit le message 1 quatre secondes après que **MMS\_New\_Session** ait été chargé. On ferme ainsi le mode édition de **MMS\_New\_Session** quand **MMS\_New\_Session** est ouvert en tant que **Template**. Le délai de quatre secondes est nécessaire. Si l'objet **pipe** n'était pas là pour produire ce délai, notre programme **thispatchlock.js** serait exécuté trop tôt pour que Max puisse en tenir compte au moment du démarrage de **MMS\_New\_Session** en tant que **Template**.

Au milieu de la copie d'écran ci-dessus vous pouvez trouver les messages **window title**, **window exec** et **window notitle**, **window exec**. Ces messages sont activés à chaque démarrage d'une session **MMS** ou lorsque l'utilisateur clique sur l'objet **toggle** (le petit carré précédemment cité). Ces messages permettent de cacher plusieurs choses qui prennent inutilement de la place sur l'écran de l'ordinateur. Il s'agit de la barre de titre de chaque fenêtre ainsi que de tous les boutons qu'elle comporte. En cachant cette barre de titre, l'utilisateur ne peut pas déplacer les fenêtres dans l'écran et leurs positions géographiques sont ainsi fixées. Cela permet également de faire apparaître ou disparaître les ascenseurs de chacune des fenêtres qui reçoit ces messages. Tous les patchers qui constituent le programme **MMS** reçoivent automatiquement ces messages à l'exception de la fenêtre de montage qui, elle, doit garder ses ascenseurs fonctionnels. Je reviendrai sur ce point en temps voulu.

Toujours dans cette même fenêtre, en haut à gauche cette fois-ci, se trouve un menu déroulant et sous ce menu déroulant se trouve un objet **patcher**. Petit rappel : l'objet **patcher** contient lui-même un patcher. Cet objet peut prendre un argument qui dans ce cas devient le nom du patcher qu'il contient. Cet argument n'est cependant pas nécessaire. En double cliquant sur cet objet **patcher**, j'ouvre son contenu et celui-ci contient un autre patcher qui lui-même va charger et/ou créer la mémoire propre à chacune des différentes sessions créées par l'utilisateur.

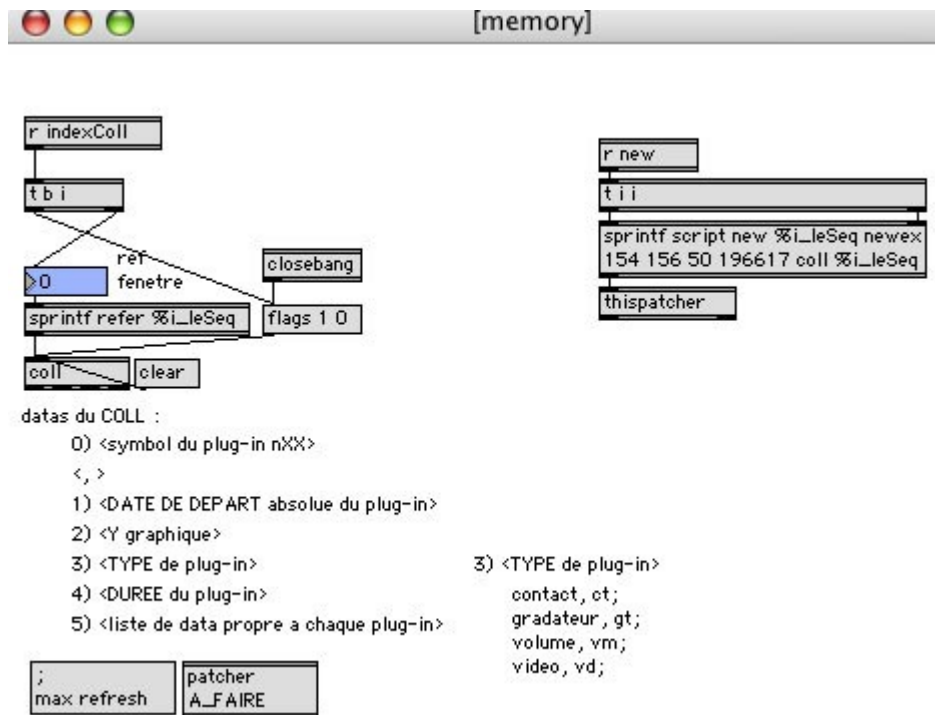
#### VII.4.3.1- Une seule et même mémoire par session

Le principe de la mémoire dans **MMS** fonctionne comme ceci : quand l'utilisateur ouvre **MMS\_New\_Session.pat** en tant que **Template**, cela revient à créer une nouvelle session de montage dans **MMS**. La session qui s'ouvre ainsi est totalement vierge. Dans la fenêtre que voici, l'utilisateur va pouvoir créer une nouvelle session en entrant le numéro de son choix dans la boîte nombre rouge.



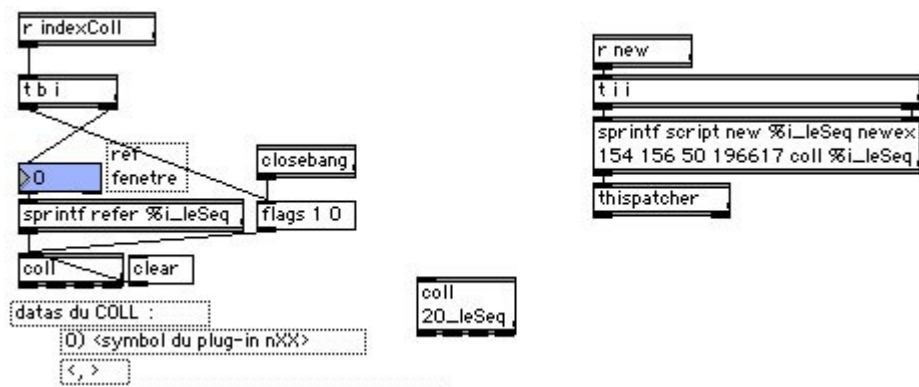
(MMS\_New\_Session\_Closed.jpg)

Par exemple, le numéro 20. Cela va créer la session de travail numéro 20. Pour entrer un numéro dans cette boîte, il doit cliquer sur le message *create a new session* juste au-dessus de la boîte. Une petite fenêtre s'ouvre alors lui permettant d'y saisir le numéro de session qu'il veut. Une fois le numéro saisi, l'utilisateur ferme cette fenêtre de saisie et clique une fois sur le numéro qui apparaît maintenant dans la boîte rouge. Ce numéro est envoyé au sous-patcher contenu dans l'objet **patcher** situé sous le menu déroulant afin de créer l'objet **coll** qui contiendra la mémoire propre à la session numéro 20 (voir copie d'écran MMS\_New\_Session\_Opened.jpg en page précédente).



(coll\_memory.jpg)

Le numéro 20 arrive dans ce patcher via l'objet **receive new** (r new) ci-dessus. L'objet **trigger** quant à lui distribue ce numéro 20 droite à gauche. Les deux sorties de **trigger** servent à transmettre la valeur 20 aux deux variables du script Max suivant : `sprintf script new %i_leSeq newex 154 156 50 196617 coll %i_leSeq` où %i est la variable de type entier qui reçoit le numéro 20. Ce script va alors créer un objet **coll** dont l'argument sera 20\_leSeq. Voici un exemple après réception du numéro 20 par le script :



(coll\_memoryCreated.jpg)

Une fois que l'objet **coll** correspondant au numéro de session entré dans la boîte rouge à été créé, l'utilisateur doit maintenant saisir dans la boîte nombre de couleur verte, le même numéro que celui déjà présent dans la boîte nombre rouge. L'utilisateur clique sur le message open situé au dessus de la boîte nombre verte, saisit le numéro de la session en question (dans notre exemple le numéro 20), puis, une fois ce numéro affiché dans cette boîte nombre, l'utilisateur clique une fois dessus. Le numéro 20 arrive alors dans le patcher présenté sur la copie d'écran ci-dessus grâce à l'objet **receive** indexColl (**r** indexColl) et est envoyé dans un objet **trigger**. A réception de ce numéro, l'objet **trigger** effectue deux tâches :

- sa sortie droite envoie le numéro de la session à un message refer afin de signaler à tous les objets **coll** devant modifier le contenu de la mémoire de la session en cours, de pointer sur le contenu de l'objet **coll** créé par script (dans notre exemple **coll 20\_leSeq**)
- sa sortie gauche permet d'envoyer un message flags 1 0 à l'objet **coll** afin que son contenu soit enregistré comme faisant partie du patch et non comme fichier externe

Avec cette technique de pointeurs multiples sur un seul et même fichier mémoire par session, nous sommes bien loin de la version interf\_proviNew.pat vue plus tôt où l'un des problèmes majeurs était justement que les données enregistrées en mémoire étaient réparties sur trop d'objets **coll** différents.

Quand l'utilisateur voudra ré-ouvrir cette session MMS numéro 20, il double-cliquera sur la session enregistrée sous un nom de son choix dans le dossier My\_Sessions. La session MMS sélectionnée s'ouvrira grâce à Max-MSP. Il faudra ensuite impérativement commencer par cliquer sur le numéro contenu dans la boîte nombre verte, puis la session sera prête à être modifiée.

Dans cette version finale, telle qu'elle est présentement, toutes les différentes données sont stockées en mémoire selon un ordre bien défini comme il est indiqué en commentaire sur la copie d'écran en page précédente. Voici cet ordre :

- symbole du *plug-in* nN, par exemple n1 comme c'était déjà le cas dans la version 060325.pat décrite précédemment.
- date absolue de départ du *plug-in*, c'est à dire sa date de début en millisecondes par rapport au pointeur de lecture du fichier

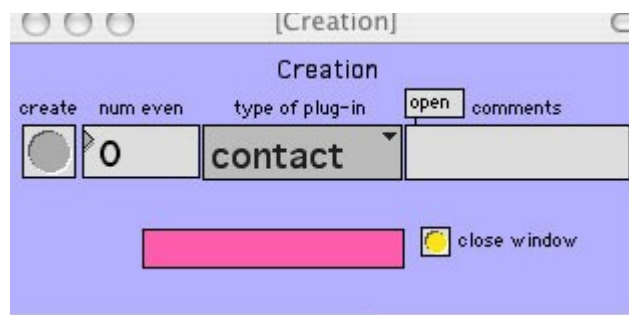
son qui, je le rappelle, est l'élément maître qui donne l'horloge de séquençement des divers *plug-ins*

- adresse graphique du *plug-in* en Y
- type de *plug-in*. Je rappelle également qu'ils sont au nombre de quatre (contact, gradateur, vidéo et volume, dans le cas d'un son)
- durée du *plug-in*
- liste de données propres à chaque *plug-in*

Avec toutes ces données enregistrées dans un ordre bien défini, il sera ensuite relativement aisé de modifier chacune de ces différentes caractéristiques indépendamment des autres comme nous le verrons plus tard lors de la description du fonctionnement interne des *plug-ins*.

#### VII.4.4- Description de la fenêtre de création de *plug-ins*

Voici une copie d'écran de la fenêtre qui va permettre à l'utilisateur de créer ses *plug-ins*. C'est la fenêtre *Creation*.



(creation\_closed.jpg)

Cette fenêtre *Creation* est minimaliste car elle se veut facile d'utilisation mais elle cache tout un programme bien plus complexe en arrière plan. Commençons par son interface utilisateur.

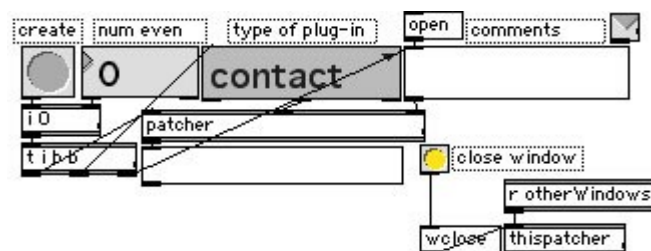
En haut à droite, vous reconnaissez le message *open* qui se trouvait déjà dans la fenêtre principale MMS. Ce même message, envoyé dans un objet **message box** (boîte message) permet d'en éditer le contenu. Quand l'utilisateur clique sur ce message, une fenêtre de saisie apparaît et permet d'éditer le contenu de l'objet **message box** auquel le message *open* est relié. L'utilisateur affecte de cette manière un commentaire (sans espace ni caractère spécial) au *plug-in* qu'il désire créer. Ce commentaire doit être explicite quant à la fonction du *plug-in* auquel il est destiné car il fait office d'aide mémoire visuelle pour l'utilisateur. C'est en lisant ce commentaire qu'il sera possible ensuite de distinguer quel *plug-in* contrôle la sortie audio numéro 1 et quel *plug-in* contrôle la sortie audio numéro 2 par exemple. Afin que mon explication soit parfaitement claire, ce commentaire pourrait être, par exemple, *audio\_out1*. Ceci dans le but de signaler à l'utilisateur que ce *plug-in* est destiné au contrôle de la sortie audio numéro 1.

Au milieu de la fenêtre *Creation*, sous le commentaire *type of plug-in*, l'utilisateur va, bien entendu, pouvoir sélectionner le type de *plug-in* qu'il désire créer. Pour ce faire, l'utilisateur doit cliquer sur le menu déroulant, maintenir le bouton de la souris enfoncé, se déplacer sur le type de *plug-in* qu'il désire créer et enfin relâcher le bouton de la souris afin de sélectionner le type de *plug-in* à créer. Le menu déroulant affiche le type contact par défaut mais une fois déroulé, vous avez le choix entre les quatre types suivants :

- *contact* (destiné à du contrôle « tout ou rien »)
- *lightDimmer* (destiné à du contrôle continu, pour des lumières de type halogène par exemple)
- *video*
- *soundLoudSpeaker* (destiné au contrôle des différentes sorties audio)

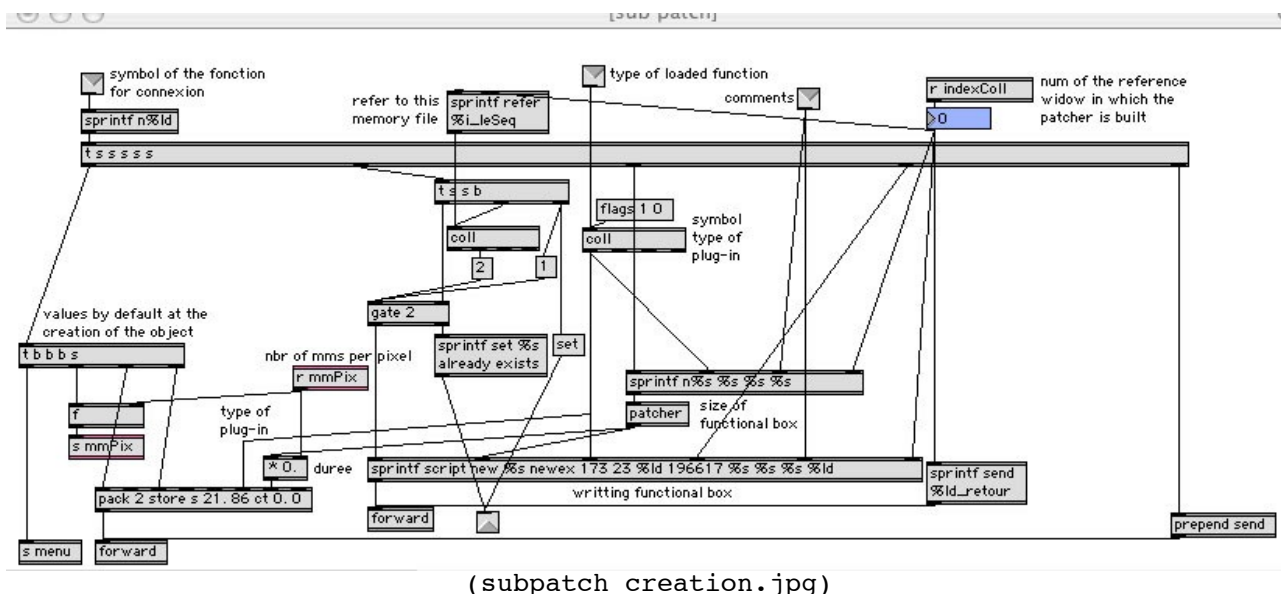
Le dernier paramètre à définir est le numéro du *plug-in*. Pour ce faire, l'utilisateur doit saisir un nombre directement dans l'objet **number box** (boîte nombre) prévu à cet effet sous le commentaire *num even* puis valider. Attention de ne pas entrer un nombre qui soit déjà utilisé au sein de cette même session : deux *plug-ins*, même s'ils sont de type différents, ne peuvent pas partager le même numéro. Si toutefois un même nombre venait à être donné deux fois par mégarde, un message d'erreur disant qu'un *plug-in* utilise déjà ce numéro apparaîtrait dans l'objet **message box** de couleur rouge qui se trouve en bas de la fenêtre et le *plug-in* ne serait pas créé.

Enfin, dernière étape au niveau de l'interface utilisateur pour la création d'un *plug-in* : cliquer sur le bouton sous le commentaire *create*. Toutes les données de création entrées jusqu'à présent ne seront envoyées au processus de création qu'après avoir appuyé sur ce bouton. En effet un objet **trigger** séquence l'envoi de ces données selon l'ordre prévu pour le processus de création, et cela indépendamment de l'ordre dans lequel l'utilisateur les aurait spécifiées. Pour illustrer cela, voici la même fenêtre *Creation* mais cette fois-ci présentée en mode édition afin que tout objet ou connexion caché soit apparent :



(creation\_opened.jpg)

C'est dans l'objet **patcher** situé au milieu de cette copie d'écran et qui est maintenant visible que le processus de création a lieu. Voyons, en page suivante, ce que cet objet **patcher** contient.



(Désolé pour la qualité de cette dernière copie d'écran, si toutefois vous vouliez en visualiser une copie de meilleure qualité mais beaucoup plus grande, ouvrez le fichier subpatch\_creation.jpg qui se trouve dans le dossier photo du CD-Rom.)

La première donnée à arriver dans ce patcher est le numéro de session que l'utilisateur spécifie à l'ouverture d'une session existante. Souvenez-vous, dans la fenêtre MMS, il y a deux boîtes nombre :

- une rouge pour la création de la mémoire propre à la session numéro N
- une verte qui contient le même numéro mais qui, elle, permet que tous les objets **coll** de la session sachent à quel objet **coll** se référer pour modifier le contenu de la mémoire

La première donnée provient de cette boîte nombre verte et de l'objet **send** indexColl auquel elle est connectée. L'objet **receive** indexColl (**r indexColl**) situé en haut à droite de la copie d'écran en page précédente reçoit ce que l'objet **send** indexColl envoie car ces derniers partagent le même argument, à savoir indexColl. L'objet **receive** indexColl distribue immédiatement ce qu'il reçoit à tous les objets auxquels il est lui-même connecté et cela selon un ordre établi dans Max-MSP même, à savoir : droite à gauche avec une priorité de bas en haut.

Le premier objet à recevoir cette donnée est donc l'objet **sprintf** qui va formater le message **send %ld retour** qu'il contient de manière à ce que l'objet **forward** auquel **sprintf** est connecté puisse envoyer les données de création qu'il recevra ultérieurement à un objet **receive** dont l'argument correspondra au message **send** qu'il a reçu. L'objet **forward** se comporte effectivement comme un objet **send** avec la particularité qu'on peut changer l'objet **receive** auquel il s'adresse grâce au message **send** suivi du nom de l'objet **receive** auquel on veut qu'il s'adresse. Le fait d'utiliser l'objet **forward** dans ce cas précis était d'autant plus justifié que nous avons l'idée de développer une possibilité de travailler en mode multi-sessions. En mode multi-session, une seule fenêtre *Creation* devrait pouvoir permettre de créer des *plug-ins* dans plusieurs fenêtres de montage et ainsi permettre de comparer



différents montages. Le code de ce patcher est donc déjà prêt pour de futures évolutions de MMS.

Le second objet à recevoir la donnée provenant de l'objet **receive** indexColl est un objet **sprintf** dont la fonction est de construire le message de création du *plug-in* désiré au fur et à mesure qu'il reçoit les données de création.

Le troisième objet est un autre objet **sprintf** dont le rôle est de construire une liste de tous les arguments du futur *plug-in*. Cette liste sera ensuite envoyée dans le patcher auquel l'objet **sprintf** est connecté. Ce patcher a pour fonction de calculer la taille graphique minimale du *plug-in* afin que ce dernier puisse contenir et afficher tous ses arguments une fois qu'il sera positionné dans la fenêtre de montage.

Enfin, le dernier objet à recevoir la donnée provenant de l'objet **receive** indexColl est encore un objet **sprintf**. Son rôle est de communiquer à l'objet **coll** auquel il est connecté à quel **coll** n\_leSeq celui-ci doit se référer afin de contrôler si le numéro qui est donné au *plug-in* lors de sa création n'est pas déjà utilisé au sein de la session en cours. Pour cela, comme nous l'avons déjà vu, nous utilisons un message `refer %i_leSeq`.

Cette donnée transmise par l'objet **receive** indexColl à tous les objets **sprintf** que nous venons de voir a donc été envoyée par l'utilisateur au moment où celui-ci a entré le numéro dans la boîte nombre verte de la fenêtre MMS de la session sur laquelle il désire travailler, et qu'il a ensuite cliqué sur cette même boîte nombre. Tous les messages contenus dans les objets **sprintf** ont maintenant leur première variable portée à cette valeur, et ce jusqu'à ce que l'utilisateur ferme la session.

Les prochaines données à entrer dans ce patcher proviennent enfin des paramètres de création spécifiés par l'utilisateur dans la fenêtre *Creation*. Comme toujours en Max-MSP, l'ordre d'arrivée des données se fait droite à gauche comme le détaillent les paragraphes suivants

Premièrement, le commentaire explicite que l'utilisateur a donné à son *plug-in* afin de pouvoir rapidement se remémorer le rôle de ce dernier. Ce commentaire est affecté aux variables de type symbole de deux objets **sprintf**. L'objet **sprintf** qui sert à la construction du script de création du *plug-in* : le commentaire prend sa place en tant qu'argument du futur *plug-in*. Ensuite, l'objet **sprintf** qui, lui, crée la liste des arguments du *plug-in* à venir afin d'ensuite l'envoyer au patcher auquel il est connecté pour que ce dernier calcule la taille graphique minimale du futur *plug-in* lors de sa création dans la fenêtre de montage, en fonction du nombre de caractères contenus dans la dite liste.

Deuxièmement, le type de *plug-in* à créer. Ce paramètre suit le même cheminement que le précédent, à la différence qu'il est d'abord envoyé dans un objet **coll**. Cet objet **coll** contient les quatre différents types de *plug-ins* ainsi que les abréviations correspondantes. Par exemple, si c'est le type contact qui a été

sélectionné par l'utilisateur, l'objet **coll** donnera l'abréviation et correspondante en sortie. C'est ensuite cette abréviation qui sera affectée aux variables de type symbole des messages contenus dans les deux objets **sprintf**. Cela permet de créer un *plug-in* de taille graphique moindre, mais aussi de rendre la lecture des différents arguments plus rapide une fois le *plug-in* créé.

Troisièmement, le numéro de *plug-in* auquel on concatène tout de suite la lettre n afin de convertir cette donnée de type *int* (nombre entier) en type *symbol*. En effet, il n'est pas possible de nommer un objet par autre chose qu'un symbole. Une fois la conversion effectuée, le nom de l'objet est ensuite acheminé droite à gauche par les sorties de l'objet **trigger**.

La première sortie de l'objet **trigger**, la plus à droite dans le patcher, transmet sa donnée à un objet **forward** via un objet **prepend send** qui va ajouter le mot *send* à la donnée fournie par l'objet **trigger**, afin de former un message *Max send n1* par exemple. L'objet **forward** qui reçoit ce message transmettra donc les données qu'il recevra ultérieurement à un objet **receive n1** quelque part dans le patch. Comme nous le verrons plus en détail ultérieurement, nous trouverons cet objet **receive n1** à l'intérieur du *plug-in* nommé n1.

La deuxième sortie de l'objet **trigger** est destinée au script de création du *plug-in* contenu dans un des objets **sprintf**. Cette donnée est affectée à la variable correspondante de type symbole et vient s'ajouter au numéro de session comme je viens de l'expliquer.

La troisième sortie est envoyée à l'objet **sprintf** qui contient le script chargé de créer la liste des arguments du futur *plug-in*. Cette donnée étant celle qui vient compléter le script contenu dans l'objet **sprintf**, celui-ci envoie alors toute la liste des commentaires qu'il contient, à savoir n1 et cnsmdl 20, au patcher auquel il est connecté. Ce dernier va s'occuper de traduire cette liste de chaînes de caractères en une taille graphique minimale pour le *plug-in* en cours de création. Cette donnée numérique est ensuite transmise au script de création du *plug-in* ainsi qu'à un objet **pack**. Ce dernier est chargé de former la liste de données de création par défaut du *plug-in*, et de l'envoyer ensuite dans la mémoire du *plug-in* une fois que celui-ci aura été créé et sera présent dans la fenêtre de montage. Je reviendrai sur ce point lors de la description des différents *plug-ins*.

La quatrième sortie est, elle, destinée à l'objet **coll** qui va vérifier, dans la mémoire de la session en cours, si l'objet de nom n1 existe déjà. Si c'est le cas, un message d'erreur apparaîtra dans la fenêtre *Creation*. L'utilisateur devra alors changer le nom du *plug-in* qu'il désire créer. Si au contraire n1 n'est pas encore utilisé, alors tout va bien : le message de création *script new n1 newex 173 23 85 196617 ct n1 cnsmdl 20* peut être envoyé, et le processus de création est achevé. Le *plug-in* apparaît alors dans la fenêtre de montage.

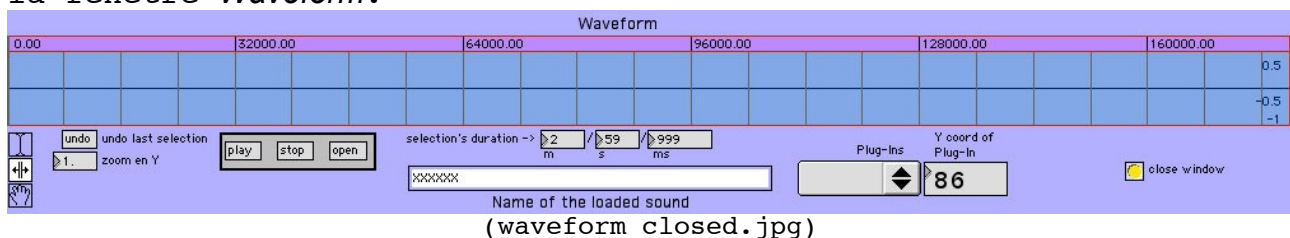


La cinquième et dernière sortie de l'objet **trigger** envoie finalement notre donnée d'exemple n1 à l'objet **pack** via un autre objet **trigger**. L'objet **pack** reçoit la variable de type symbole attendue (dans notre exemple n1), tout de suite suivie d'un message bang qui permet d'envoyer toute la liste des données de création par défaut de notre *plug-in* n1 à l'objet **coll** contenu dans ce même *plug-in*. C'est à l'aide de cet objet **coll** que toutes les données propres à n1 seront enregistrées dans la mémoire de la session à laquelle n1 appartient désormais (dans notre exemple la session 20 qui utilise l'objet **coll** 20\_leSeq). Enfin, n1 est ajouté dans un menu déroulant qui se trouve dans la fenêtre *Waveform* : la fenêtre dans laquelle la forme d'onde du fichier audio utilisé par la session en cours est représentée graphiquement.

#### VII.4.5- Description de la fenêtre Waveform (forme d'onde)

Lors de l'ouverture de toute session MMS, plusieurs fenêtres apparaissent à l'écran. Ces fenêtres constituent la partie que l'utilisateur sera le plus souvent amené à utiliser au sein de l'interface graphique de l'application MMS. La fenêtre *Waveform* y tient une place d'importance, car c'est dans cette fenêtre que le fichier audio qui contrôle le séquençement des événements est représenté de manière graphique. C'est également grâce à cette fenêtre que l'utilisateur va définir la durée mais aussi l'adresse de déclenchement des divers *plug-ins*, ainsi que leur positionnement en Y (verticale) par un simple procédé de sélection couplé à l'utilisation d'un menu déroulant. L'interface graphique est extrêmement simple d'utilisation, ce qui est le résultat d'un effort de programmation particulier. Je vais faire de mon mieux pour n'en expliquer que l'essentiel tout en vous permettant de comprendre l'ensemble des divers patchers qui la constituent.

Commençons par regarder rapidement l'interface graphique de la fenêtre *Waveform*.

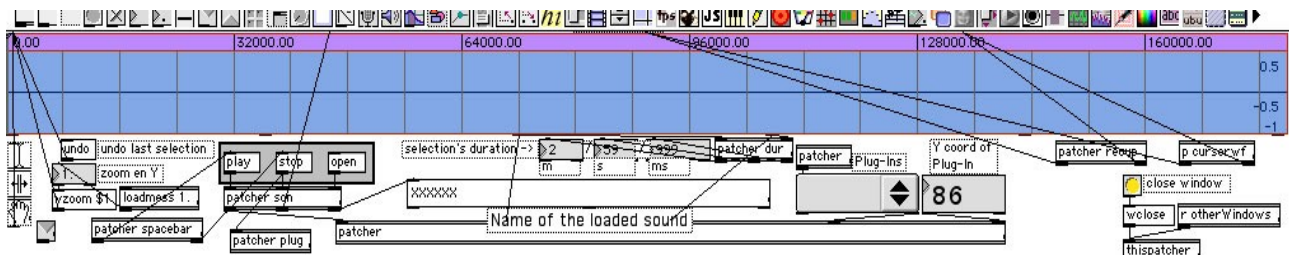


Tout d'abord, un axe de temps qui, par défaut, s'étale sur trois minutes (180.000 millisecondes) mais qui s'adapte automatiquement à la longueur du fichier audio qu'il affiche. Ensuite, droite à gauche, trois outils :

- le premier en partant du haut permet à l'utilisateur d'effectuer une sélection au sein du fichier audio
- celui du milieu permet de déplacer une certaine sélection gauche à droite, mais surtout d'en changer la taille suivant que l'utilisateur maintient le bouton de la souris enfoncé et la fait glisser vers le haut (élargissement de la sélection) ou vers le bas (rétrécissement de la sélection)
- le dernier outil permet de zoomer sur l'axe du temps (X)

On voit ensuite un bouton qui permet d'appliquer la fonction undo (annuler) mais uniquement sur la dernière sélection effectuée et une boîte nombre pour zoomer sur l'axe d'amplitude (Y) du fichier audio affiché dans la fenêtre *Waveform*. Suivent les trois boutons *play*, *stop* et *open*. Les deux premiers sont affectés à la mise en route et à l'arrêt de la lecture du fichier audio. Le troisième, *open*, sert à ouvrir une fenêtre de dialogue qui permet de choisir le fichier audio à charger. Au milieu de la fenêtre *waveform*, se trouve un compteur qui affiche la longueur de la sélection en cours au format minutes / secondes / millisecondes. Au-dessous de ce compteur apparaît le nom du fichier audio chargé. Quand la fenêtre *Waveform* n'a aucun fichier audio à afficher, le nom du fichier audio est remplacé par le symbole XXXXX. Il se trouve ensuite un menu déroulant. Quand l'utilisateur clique sur celui-ci, il affiche le nom de tous les *plug-ins* contenus dans la session. Sélectionner le nom d'un *plug-in* dans ce menu permet d'en changer les paramètres dont, par exemple, sa position en Y dans le banc de montage. Cette coordonnée en Y peut être changée à l'aide de la boîte nombre située à droite de ce menu déroulant.

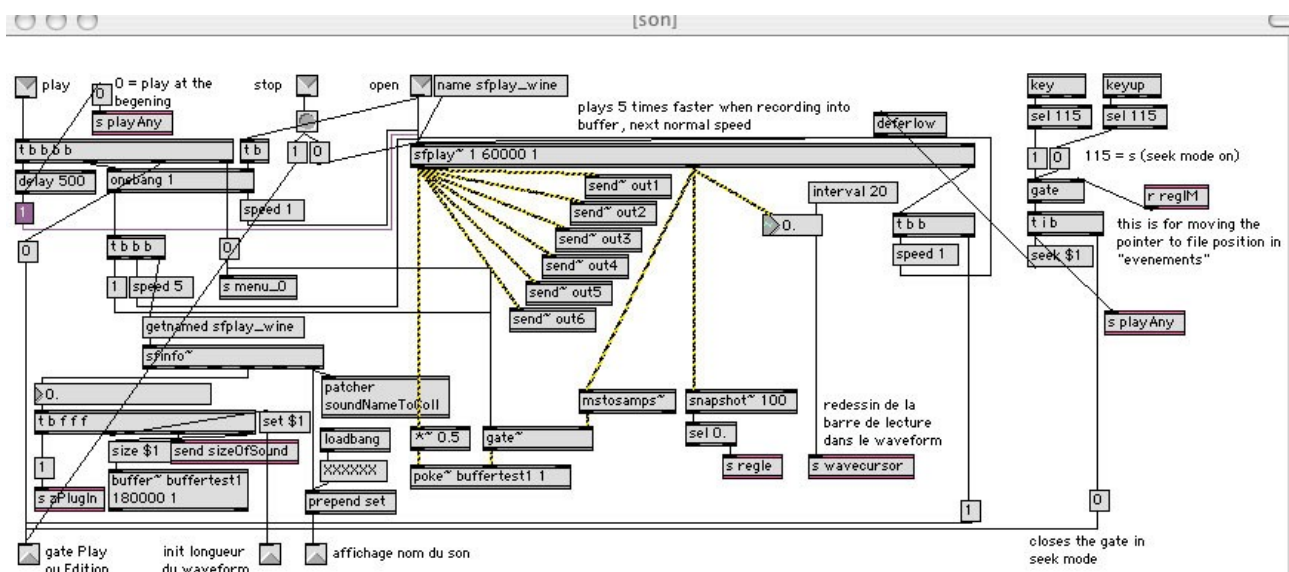
Comme vous pouvez le voir sur la copie d'écran ci-dessous, derrière cette interface très simple se cachent une multitude de patchers plus ou moins complexes.



(waveform\_opened.jpg)

Là encore, pour une meilleure visibilité, vous pouvez ouvrir le fichier *waveform\_opened.jpg* situé dans le dossier photo du CD-Rom.

Commençons par regarder ce que contient l'objet **patcher son**.



(MMS\_son.jpg)

Je n'expliquerai pas tout immédiatement car je serai amené, de toute façon, à revenir sur certaines parties lors de l'explication d'autres patchers.

Lorsque vous cliquez sur le message open dans le but de charger un son dans votre session MMS, cela ne fait finalement qu'envoyer le message open à l'objet **sfplay~**. C'est cet objet qui va aller lire, directement sur votre disque dur et non en mémoire RAM, le fichier que vous désirez utiliser. Ce message accède au patcher son via l'objet **inlet** (entrée) en haut, au milieu du patcher. Le message open est également converti en un message bang lors de son passage par l'objet **t b** (**trigger** bang). Cet objet **trigger** envoie aussitôt un message bang à l'entrée droite de l'objet **onebang** auquel il est connecté. De cette manière, lorsque l'objet **onebang** recevra un message bang dans son entrée gauche, celui-ci enverra un même message bang par sa sortie gauche. Tout prochain message bang reçu par l'objet **onebang** dans son entrée gauche passera ensuite par sa sortie droite jusqu'à ce qu'un autre message bang soit envoyé à son entrée droite et ainsi de suite... De cette manière, quand l'utilisateur clique sur le message play dans la fenêtre *Waveform*, l'objet **onebang** va envoyer un message bang par sa sortie droite. Ce message sera distribué à une série d'objets via un objet **trigger**. Le premier message bang à être envoyé par l'objet **trigger** est destiné à l'objet MSP **sfinfo~**. Cet objet va permettre de récupérer des informations concernant le fichier audio chargé dans **sfplay~** comme par exemple son nom et sa durée. Le nom pourra ainsi être affiché dans la fenêtre *Waveform*. Quant à l'information de durée, elle va permettre à l'axe du temps de la fenêtre *Waveform* de s'adapter à la longueur du fichier audio choisi. Un autre message bang envoyé par l'objet **trigger** va, quant à lui, activer une fonction de lecture accélérée pour l'objet **sfplay~** (speed 5) afin que celui-ci joue le fichier audio choisi par l'utilisateur cinq fois plus vite. La raison pour laquelle le fichier audio est lu cinq fois plus vite lors de sa première lecture et qu'il va falloir écrire ce fichier audio dans un objet **buffer~** afin que sa forme d'onde puisse être rendue visible dans la fenêtre *Waveform*. En effet, l'objet **buffer~** situé dans le patcher son et l'objet **waveform~** qui a donné son nom à la fenêtre dans laquelle il se situe partagent leurs informations. Jouer le fichier audio cinq fois plus vite permet ainsi de gagner du temps. La prochaine fois que l'utilisateur cliquera sur le message play, l'objet **sfplay~** recevra un message speed 1 ; 1 étant la valeur de vitesse de lecture normale. Enfin, quand l'utilisateur clique sur le bouton stop, le message est aussitôt converti en la valeur 0. Quand l'objet **sfplay~** reçoit le nombre zéro dans son entrée gauche, il cesse instantanément de jouer.

Que se passe-t-il en sortie de l'objet **sfplay~** maintenant ? Notre objet **sfplay~** compte trois sorties. Étudions en les deux dernières toujours en respectant le séquençement selon Max-MSP, c'est à dire de la droite vers la gauche.

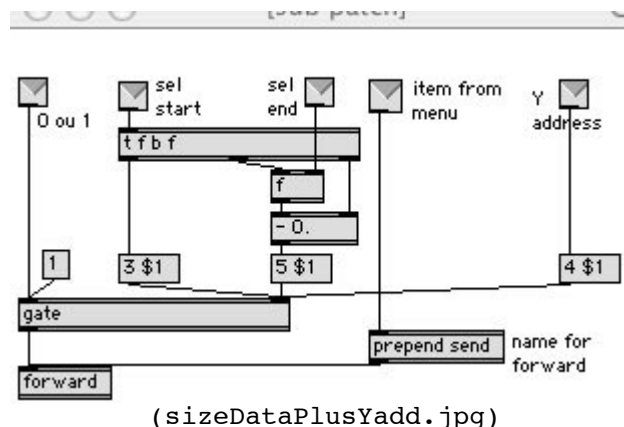
La sortie du milieu de l'objet **sfplay~** dans le patcher nommé son est une sortie qui donne des informations sur le fichier en cours de lecture. Ces informations sont au format signal et correspondent à la position du pointeur de lecture de **sfplay~** en

millisecondes. J'utilise ces données pour contrôler les deux pointeurs de lecture de MMS, à savoir : un pointeur dans la fenêtre *Waveform* et un second dans la fenêtre de montage. Il est évident que les deux pointeurs d'une session MMS doivent être synchronisés afin que l'utilisateur puisse faire le lien visuel entre ce qu'il voit dans la fenêtre *Waveform* et ce qu'il voit dans la fenêtre de montage.

La sortie la plus à gauche est la sortie audio de l'objet **sfplay~**. Elle est, bien entendu, envoyée aux six différentes sorties audio de l'application MMS. Elle est également envoyée dans un objet **poke~** afin que le fichier audio lu par l'objet **sfplay~** soit enregistré dans la mémoire d'un objet **buffer~** de même argument que **poke~** (ici `buffertest1`). Cette opération a pour but de permettre à l'objet **waveform~** situé dans la fenêtre *Waveform* d'afficher la forme d'onde du fichier audio avec lequel l'utilisateur désire effectuer et séquencer son montage. Au passage, on peut voir que le signal de la deuxième sortie de **sfplay~** est également envoyé dans la seconde entrée de notre objet **poke~**. Cette entrée permet de spécifier l'adresse (en millisecondes) à laquelle doivent être enregistrés les échantillons du signal audio acheminé dans l'entrée gauche de l'objet **poke~**.

#### VII.4.5.1- Édition des données propres à un *plug-in* particulier

Dans la fenêtre *Waveform*, sous l'affichage de la durée de sélection, se trouve également un autre objet **patcher**. C'est dans ce patcher que les données propres à un *plug-in* particulier vont pouvoir être calculées et/ou modifiées. Elles seront ensuite « labélisées » afin de pouvoir être identifiées ultérieurement. C'est également à partir de ce patcher qu'elles seront enfin envoyées au *plug-in* auquel elles sont destinées. Étudions ce patcher.

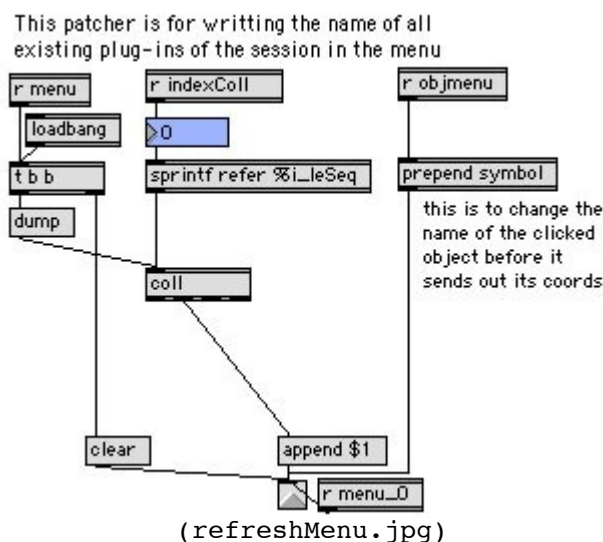


Quoi que l'utilisateur veuille effectuer comme opération sur un *plug-in* existant, il doit tout d'abord commencer par le sélectionner dans le menu déroulant qui se trouve dans la fenêtre *Waveform*. Le nom du *plug-in* en question entre dans le patcher par l'entrée décrite par le commentaire *item from menu* et est alors affecté à l'objet **forward** qui se trouve en bas de la copie d'écran ci-dessus. A partir de ce moment, l'objet **forward** enverra au *plug-in* spécifié chacune des données qu'il recevra de l'objet **gate** auquel

il est connecté. L'adresse de départ du *plug-in* est labélisée par le chiffre 3, elle correspond au début de la sélection dans la forme d'onde qui apparaît dans la fenêtre *Waveform*. La durée du *plug-in* est d'abord calculée. Pour cela on soustrait l'adresse de début de sélection à l'adresse de fin de sélection. Le résultat est quant à lui labélisé par le chiffre 5. Pour ce qui concerne l'adresse en Y du *plug-in*, celle-ci est labélisée par le chiffre 4. Si par exemple l'utilisateur décide de changer l'adresse en Y du *plug-in* de nom n1, il devra sélectionner le *plug-in* n1 dans le menu déroulant, ce qui aura pour effet de commander à l'objet **forward** d'envoyer les données qu'il va recevoir de l'objet **gate** à un objet **receive** n1 quelque part dans le patch (en l'occurrence, dans le *plug-in* n1). Il pourra alors changer la valeur dans la boîte nombre située à droite du menu déroulant dans la fenêtre *Waveform* pour une nouvelle valeur de son choix. Cette valeur sera mise dans une liste dont le premier élément sera le chiffre 4, et le tout sera envoyé à l'objet **receive** n1.

#### VII.4.5.2- Sélection d'un *plug-in* pour édition

J'ai déjà été amené à parler du menu déroulant de la fenêtre *Waveform* à plusieurs reprises. Voyons ce qu'il contient et comment il fonctionne.



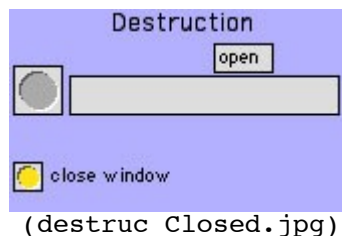
Une fois de plus, le patcher utilise le numéro de session qui est envoyé à un message `refer %i leSeq`. Vous êtes maintenant familiarisé avec cette technique. L'objet **coll** auquel ce message s'adresse pointe donc sur le contenu de l'objet **coll** principal propre à la session en cours. Chaque fois qu'un *plug-in* est créé ou effacé, le fichier de mémoire de la session est automatiquement et immédiatement mis à jour, ce qui signifie que le contenu de l'objet **coll** s'y référant l'est également. A chaque création ou destruction de *plug-in*, ce patcher reçoit un message `bang` via l'objet **receive** menu (**r menu**), ce qui a pour effet de faire envoyer un message `clear` à l'objet **umenu** (menu déroulant) via un objet **trigger**. Ce message `clear` efface le contenu total de l'objet **umenu** mais ce dernier est automatiquement remis à jour grâce au message `dump` aussitôt envoyé à l'objet **coll**. Ce dernier envoie

alors immédiatement à l'objet **umenu** le nom de tous les *plug-ins* existants dans la session en cours. Notre menu déroulant contient ainsi à nouveau la liste complète de tous les *plug-ins* présents dans la fenêtre de montage.

Il reste encore un patcher important à regarder de près dans cette fenêtre *Waveform*. Ce patcher est le **patcher** plug. C'est dans ce patcher que se fait la lecture du contenu du fichier mémoire de la session et donc l'interprétation du montage réalisé. Cependant, le **patcher** plug est le dernier patcher que nous étudierons car pour bien en comprendre le fonctionnement, il est préférable de cerner comment fonctionnent tous les autres patchers qui constituent l'application MMS ainsi que les différents *plug-ins*. Je reviendrai donc sur ce patcher plus tard.

#### **VII.4.6- Explication de la fenêtre Destruction**

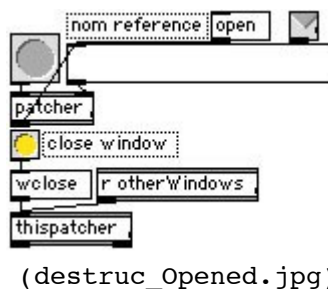
La fenêtre de destruction est très facile à cerner aussi bien au niveau de l'interface utilisateur que de la programmation qui la constitue. Voici une copie d'écran de la fenêtre destruction en mode fermé :



Le rôle de la fenêtre destruction, comme son nom l'indique, est de détruire un ou plusieurs *plug-in(s)* à la fois afin qu'il(s) ne fasse(nt) plus partie de la session et qu'il(s) n'apparaisse(nt) donc plus dans la fenêtre de montage. Pour spécifier à la fenêtre destruction le numéro du(des) *plug-in(s)* qu'elle doit faire disparaître de la session en cours, il faut procéder de la même façon qu'avec toutes les autres fenêtres de l'application MMS, c'est à dire :

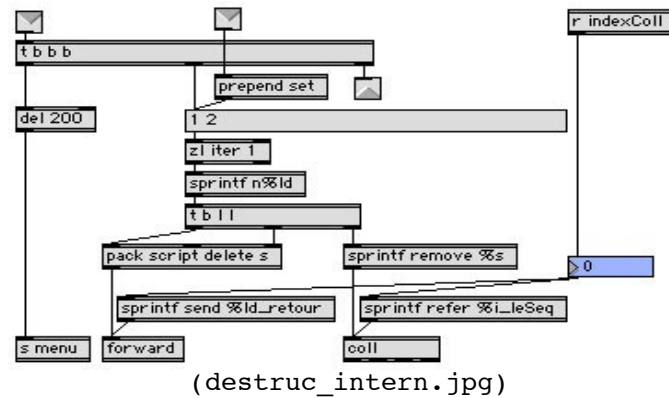
- cliquer sur le bouton *open*. Une fenêtre de saisie apparaît dans laquelle l'utilisateur saisit le numéro du ou des différents *plug-in(s)* qu'il désire détruire (ex. : 1 5 9). Attention, il ne faut saisir que le numéro d'un *plug-in* et non pas son nom. Autrement dit, il ne faut pas saisir, par exemple : n1 n5 n9
- fermer ensuite la fenêtre de saisie
- cliquer sur le bouton situé à gauche de la liste de *plug-ins* à détruire

Examinons rapidement le fonctionnement de la fenêtre Destruction.





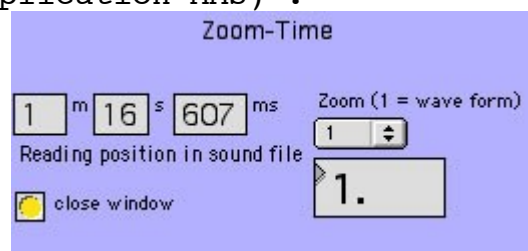
Quand l'utilisateur clique sur le bouton de la fenêtre Destruction, la liste des *plug-ins* à détruire est envoyée à l'objet **patcher**. Celui-ci contient le patcher suivant :



Comme vous pouviez vous y attendre, on retrouve encore une fois notre numéro de session, le message refer %i\_leSeq et bien sûr l'objet **coll**, car nous allons modifier le contenu de la mémoire de la session par l'effacement de certains éléments (certains *plug-ins*). La liste des numéros de *plug-ins* à détruire arrive ensuite dans le patcher par l'entrée du milieu. L'objet **iter** possède l'argument 1 ce qui signifie qu'il va découper la liste entrante en tranches ne comprenant plus qu'un seul élément jusqu'à ce que la liste soit vide. L'objet **sprintf** n%\_leSeq, quant à lui, concatène la lettre n au numéro qu'il reçoit afin de recréer le nom des divers *plug-ins* à détruire (ex : n1 n5 n9). A l'aide de l'objet **sprintf** remove %s, on met en forme un message remove où %s est la variable qui correspond aux différents noms de *plug-ins* qu'il va falloir effacer de la mémoire de la session. Ces messages remove sont envoyés à l'objet **coll** et les différents *plug-ins* à détruire sont effacés de la mémoire de la session. Le message delete qui suit efface graphiquement le ou les *plug-in(s)* dans la fenêtre de montage. Enfin, le menu de la fenêtre *Waveform* est également mis à jour afin que seuls les *plug-ins* présents dans la session y apparaissent.

#### VII.4.7- Description de la fenêtre de zoom (Zoom-Time)

Il y a peu à dire sur la fenêtre de zoom tant elle est simple en terme d'interface utilisateur. Sa partie programmation est elle aussi relativement simple, même si elle est le résultat de longues recherches. Toute la complexité de la fonction de zoom se trouve dans la fenêtre de montage que nous étudierons juste après. Pour l'instant, voici une copie d'écran de la fenêtre de zoom (nommée Zoom-Time dans l'application MMS) :



Je n'en montrerai qu'une version fermée tant la programmation qui se cache derrière l'interface se limite à quelques objets.

Sur la gauche se trouve un affichage de la position du pointeur de lecture de l'objet **sfplay~** (vu dans le **patcher son**) lors de la lecture du fichier audio chargé pour la session. Les données envoyées au format millisecondes par le **patcher son** sont converties par des calculs arithmétiques afin d'en afficher un format plus lisible en minutes, secondes et millisecondes.

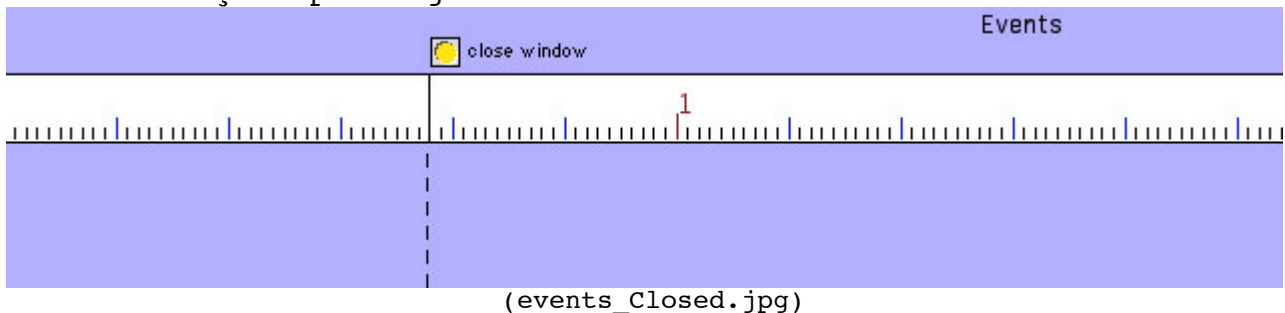
Viennent ensuite deux possibilités de définir le facteur de zoom à utiliser :

- un menu déroulant qui contient des valeurs de zoom pré-définies (des *presets*)
- une boîte nombre qui permet à l'utilisateur de spécifier un autre facteur de zoom, bien que ceux définis dans le menu soient recommandés

Facteur 1 = pas de zoom, facteur 2 = deux fois plus petit, facteur 0.5 = deux fois plus grand... etc. La valeur de zoom choisie est envoyée à la fenêtre de montage via un simple objet **send**.

#### VII.4.8- Description de la fenêtre de montage (*Events*)

La fenêtre de montage, nommée *Events* dans le patch, est la fenêtre dans laquelle apparaissent les *plug-ins* une fois créés. Elle est différente des autres fenêtres de l'application MMS car c'est la seule fenêtre dont les ascenseurs sont présents. Ceux-ci sont indispensables car cette fenêtre offre une fonction particulièrement intéressante de zoom. La fenêtre de montage et la fenêtre *Waveform* sont étroitement liées car la taille de l'objet **waveform~** situé dans la fenêtre *Waveform* est l'une des composantes d'une équation qui permet de déterminer le nombre de millisecondes représentées par pixel dans la fenêtre de montage, et ce en fonction du facteur de zoom en vigueur. Cette fenêtre est le résultat d'une longue période de recherche car, quand l'utilisateur modifie le facteur de zoom, ce n'est pas la taille des objets qui change mais la résolution de la fenêtre de montage. Mais commençons par regarder le côté interface utilisateur.

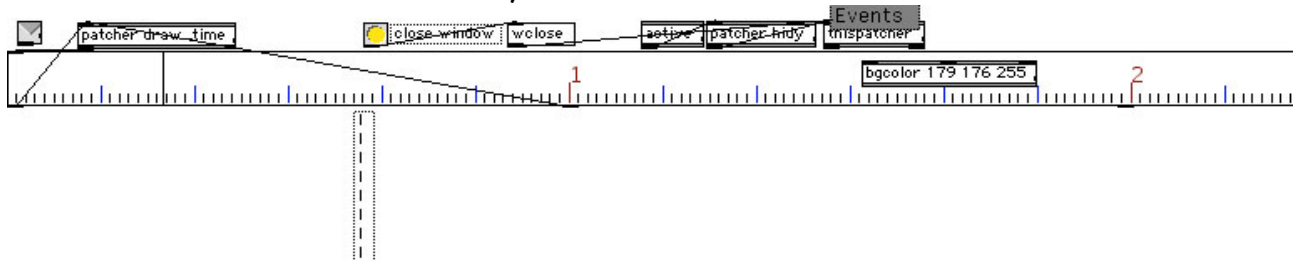


Voici une copie d'écran d'une partie de la fenêtre de montage (la fenêtre *Events*). Sur cette copie d'écran, la fenêtre est vierge. Elle contient un axe temps qui, par défaut, affiche une durée de trois minutes. Pourquoi trois minutes ? Tout simplement parce que cela dépend de la longueur du fichier audio chargé dans la session et que, comme nous l'avons vu au tout début de la partie VII.4.5, quand l'objet **buffer~** du **patcher son** est vide, l'objet **waveform~** affiche par défaut 180.000 millisecondes, donc trois minutes. Les deux fenêtres étant liées, la fenêtre montage affiche également trois minutes. Vous pouvez également voir le



pointeur de lecture de la fenêtre de montage. Ce pointeur est synchronisé avec le pointeur de l'objet **waveform~**, tous deux contrôlés par les données envoyées par la deuxième sortie de l'objet **sfplay~** comme nous l'avons vu précédemment.

Voici la même fenêtre, mais ouverte :



(events\_Opened.jpg)

En haut à gauche de la copie d'écran ci-dessus, apparaît désormais un objet **patcher** avec pour argument `draw_time` (« dessiner le temps »). L'argument `draw_time` n'est pas indispensable comme je l'ai déjà dit précédemment mais le fait de donner un argument à un objet **patcher** a deux avantages :

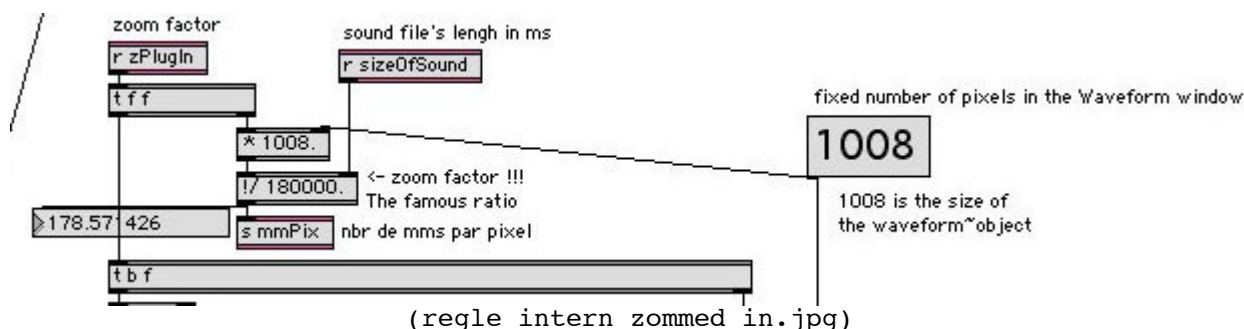
- c'est un aide mémoire pratique pour le développeur du patch
- cela a pour effet de nommer le patcher que l'objet **patcher** contient

C'est dans ce patcher `draw_time` que toute la gestion graphique et celle du facteur de zoom se déroule. Le contenu de ce patcher étant relativement conséquent, je vous invite à vous référer à la copie d'écran nommée `regle_intern` en annexe de ce mémoire, dans la partie Photos. Je n'expliquerai que les grandes lignes de son fonctionnement tout en m'efforçant d'être le plus clair possible.

Comme je le disais en page précédente, l'objet **waveform~** et la fenêtre de montage sont étroitement liés. L'objet **waveform~**, dans la fenêtre **Waveform**, est représenté avec une largeur de 1008 pixels très exactement. Cette valeur 1008 est la référence qui va permettre de définir combien de secondes sont représentées par pixel. Voilà comment le patcher procède :

- on dit que l'objet **waveform~** a une largeur de 1008 pixels
- si l'objet **waveform~** et la fenêtre de montage partagent la même durée, un taux de zoom de 1 définit alors la fenêtre de montage comme représentant également 1008 pixels
- si le taux de zoom est porté à 2, alors la fenêtre de montage représente maintenant 2016 pixels en largeur, et si le taux de zoom est porté à 0.5, la fenêtre de montage ne représentera plus que 504 pixels en largeur
- maintenant qu'on connaît la taille de la fenêtre de montage, trouvons l'échelle de son contenu : pour un pixel, combien de secondes ?

Voici une copie d'écran qui montre la partie du patcher qui calcule cela :

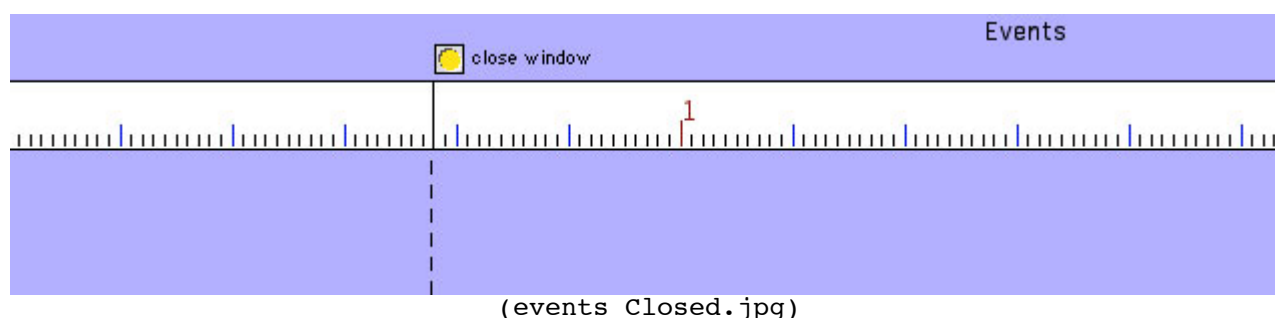


Le facteur de zoom à été envoyé à partir de la fenêtre *zoom-time* via un objet **send** zPlugIn. Comme on peut le voir sur la copie d'écran, notre patcher reçoit ce facteur de zoom via l'objet **receive** zPlugIn (**r** zPlugIn).

La durée en millisecondes du fichier audio utilisée dans la session provient, elle, d'un objet **send** sizeOfSound situé dans le patcher son. Notre patcher contient un objet **receive** de même nom et reçoit ainsi la valeur de durée du fichier audio (par défaut 180.000).

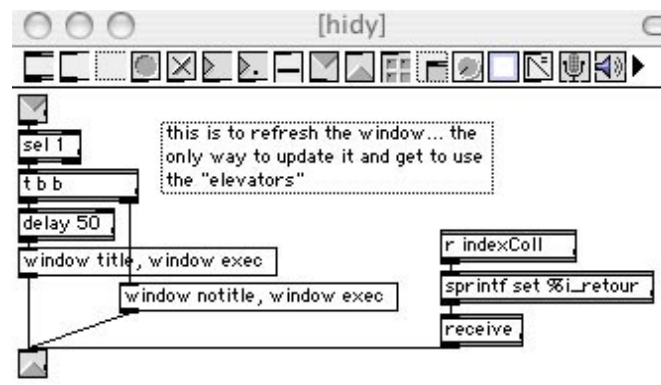
On connaît la longueur du fichier audio, on connaît également le facteur de zoom ainsi que la taille en pixels de l'objet **waveform~**. Le calcul est donc le suivant : le nombre fixe de pixels de l'objet **waveform~** (1008), multiplié par le facteur de zoom (1 par défaut), le tout divisé par la durée du fichier audio en millisecondes (180.000 par défaut). Avec les valeurs par défaut, la fenêtre de montage contient 0,0056 millisecondes par pixel. Avec un facteur de zoom de 2, le résultat est 0,0112... etc. L'immense avantage d'une telle technique est qu'elle est uniquement graphique : en aucun cas les données de départ ou de durée des divers *plug-ins* contenus dans la fenêtre de montage ne sont affectées.

Toute la partie basse du patcher *draw\_time* sert à dessiner l'axe de temps de la fenêtre de montage dans un objet **lcd**. Chaque seconde est représentée graphiquement sur l'axe de temps par un trait de couleur noir et tous les dix traits, le trait est légèrement plus grand et de couleur bleue. Chaque minute est représentée par un grand trait rouge surplombé du numéro de la minute représentée. Voici à nouveau la copie d'écran d'une partie de la fenêtre de montage afin de voir le résultat graphique :



Enfin, il ne reste dans cette fenêtre de montage qu'un petit patcher à décrire. C'est un patcher dont le nom est **hidy**. Il se situe au-dessus de l'axe du temps et est connecté à l'objet

**thispatcher** auquel il envoie ses messages. En voici une copie d'écran en mode ouvert :



(hack.jpg)

Ce patcher contient deux parties importantes. La première est la partie droite. L'objet **receive** indexColl transmet à l'objet **sprintf** le numéro de la session en cours. L'objet **sprintf**, qui contient un message set %i\_retour, va affecter l'information qu'il reçoit de l'objet **receive** indexColl à sa variable %i afin de former, par exemple, le message set 20\_retour. Ce message est alors envoyé à l'objet **receive**. Ceci revient à définir 20\_retour comme argument de l'objet **receive**. Ce dernier va donc maintenant pouvoir recevoir des données d'objets **send** 20\_retour ou d'objets **forward** ayant également reçus 20\_retour comme argument. Or, nous avons vu dans la partie VII.4.4 qu'une fois entièrement écrit, le script contenant toutes les données de création d'un *plug-in* était envoyé à un tel objet **forward**. C'est précisément dans ce patcher *hidu* que ces données arrivent et sont immédiatement transmises à l'objet **thispatcher** qui se trouve dans la fenêtre de montage afin que le *plug-in* y soit créé.

En fin de chapitre V.7.5, je disais utiliser un *hack* (une astuce) pour rafraîchir l'ambitus de déplacement des ascenseurs d'un patcher lorsqu'on y déplace un objet par script. Ce *hack* est la partie gauche du patcher *hidu*.

Dans la fenêtre de montage, se trouve un objet **active**. Cet objet a pour fonction d'envoyer un message 1 lorsque la fenêtre dans laquelle il se trouve est au premier plan. Si l'utilisateur déplace un *plug-in* en Y au-delà des limites de la fenêtre, les ascenseurs de cette dernière ne seront malheureusement pas mis à jour automatiquement et il lui sera alors délicat de visualiser le *plug-in* ainsi déplacé par script. Pour régler ce problème, j'utilise le message 1 envoyé par l'objet **active** pour rafraîchir la fenêtre de montage chaque fois que celle-ci est au premier plan. Ce message 1 arrive dans un objet **sel** qui a pour argument le nombre 1. Autrement dit, si l'objet **sel** reçoit le nombre 1 dans son entrée gauche, il va immédiatement envoyer un message bang par sa sortie gauche. Ce même message bang est à son tour envoyé dans un objet **trigger** qui va se charger d'envoyer un autre message bang aux deux messages auxquels il est connecté. Ce sont ces messages window notitle, window exec et window title, window exec qui, envoyés à intervalles de 50 millisecondes à l'objet **thispatcher**

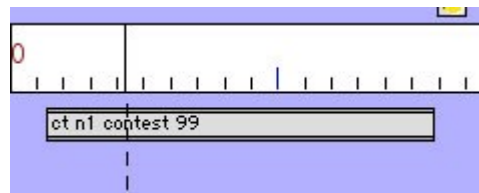
contenu dans la fenêtre de montage, vont permettre ce rafraîchissement graphique. De cette manière, Max va réactualiser les ascenseurs de la fenêtre de montage avec la position géographique des divers objets qu'elle contient. J'espère quand même qu'un jour, *Cycling'74* règlera ce problème de rafraîchissement de fenêtre lors d'une prochaine version.

#### VII.4.9- Description des différents *plug-ins*

Jusqu'à présent, nous avons vu comment les *plug-ins* sont créés dans la fenêtre de montage (*Events*) de l'application MMS. Nous avons également vu que MMS n'utilise qu'une seule mémoire par session et que celle-ci peut être éditée en divers points du patch grâce à une multitude d'objets *coll* s'y référant. Nous allons maintenant étudier ce que contiennent les différents types de *plug-ins* afin de comprendre le format de données que chacun d'entre eux écrit en mémoire et comment.

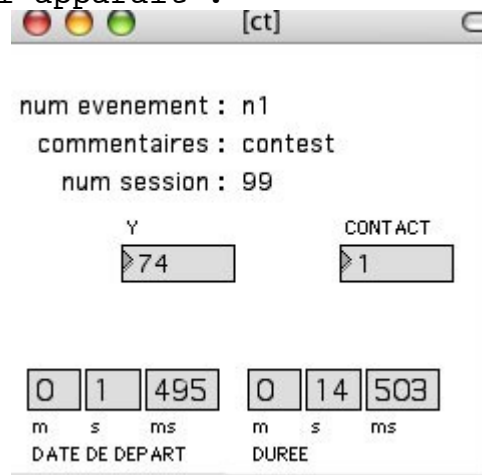
##### VII.4.9.1- Le *plug-in* de type contact

Voici l'exemple d'un *plug-in* de type contact et de nom *n1*, dont le commentaire est « contest » et qui appartient à la session numéro 99.



(ct\_eventsWindow.jpg)

Quand l'utilisateur double-clique dessus afin d'en éditer le contenu, voici ce qui apparaît :



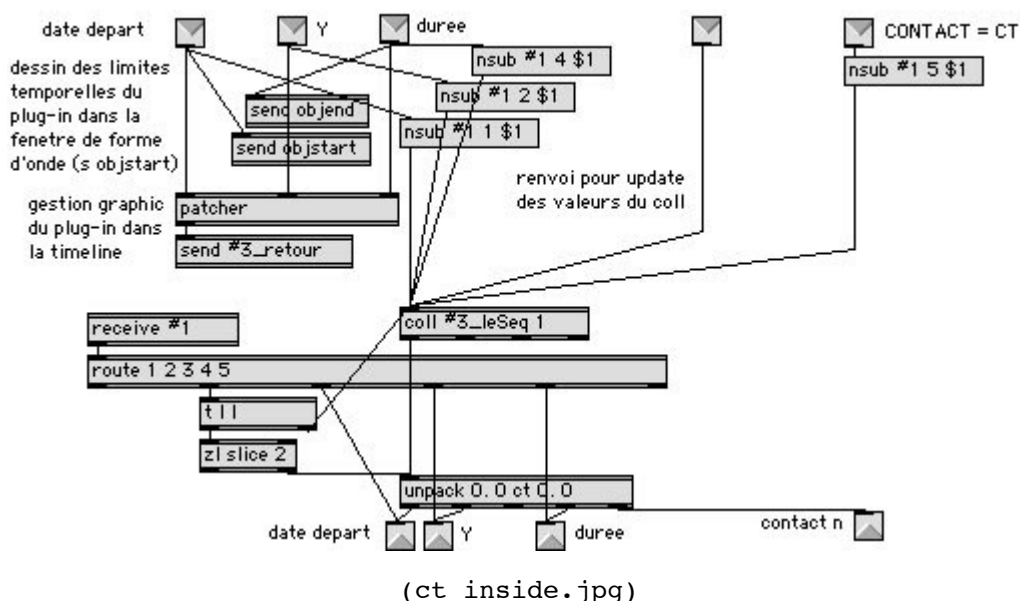
(ct\_interf.jpg)

Chacune des données présentes dans ce *plug-in* peut être modifiée. La boîte nombre sous le commentaire *Y* représente bien entendu l'adresse *Y* du *plug-in* dans la fenêtre de montage. Cette adresse peut être changée à partir de la fenêtre *Waveform*, comme nous l'avons vu précédemment, mais également à partir de la présente. Le paramètre *contact* est tout simplement le numéro de contact à contrôler. Par défaut, il me semble plus clair de lui donner le même numéro que celui de son nom mais si l'utilisateur préfère lui

donner un autre numéro, cela est possible (ici le *plug-in* se nomme n1 et son numéro est également 1, il contrôle donc le contact numéro 1). Les paramètres date de départ et durée sont normalement donnés en sélectionnant la partie du fichier son dans la fenêtre *Waveform* à laquelle le *plug-in* s'applique. Cependant, il est également possible de régler ces deux paramètres de façon plus fine directement à l'intérieur du *plug-in* en modifiant les valeurs affichées. Cela permet une précision à la milliseconde.

Comme vous pouvez le voir, l'interface est extrêmement simple. Les commentaires en haut de la fenêtre sont un aide mémoire pour l'utilisateur. Ils affichent le nom du *plug-in*, son commentaire et le nom de la session à laquelle il appartient.

Regardons maintenant la programmation interne du type de *plug-in* in contact.



Voici l'intérieur de l'abstraction *ct.pat* située dans le dossier *MMS\_lib* qui se trouve lui-même dans le dossier *MMS\_All*. Ce que le script de création a créé en réalité est une boîte objet Max quelconque, identique à celle utilisée systématiquement lors de la création d'un objet à partir de la barre d'outils d'un patcher en mode édition. La différence est que dans cette boîte objet, on a inscrit *ct n1 contest 99*. Max va alors chercher dans les chemins d'accès spécifiés par l'utilisateur (menu Option, File Preferences) s'il trouve un objet ou un patcher de nom *ct*. C'est effectivement le cas (voir chapitre VII.2) : il existe bien un patcher de nom *ct.pat* (dans le dossier *MMS\_lib*), qui sera aussitôt chargé dans notre boîte objet. C'est ce qu'on appelle une abstraction en jargon Max-MSP. Cette abstraction nécessite trois arguments :

- #1 = nom du *plug-in* (dans notre exemple n1)
- #2 = commentaire (dans notre exemple contest)
- #3 = numéro de session (dans notre exemple 99)

On retrouve #1, #2 et #3 dans la copie d'écran ci-dessous.

Quand le *plug-in* est dans la fenêtre de montage, il n'est pas possible d'accéder à cette partie du patcher car il est impossible

de mettre une abstraction en mode édition lorsque celle-ci est utilisée. Sachez donc que, dans le *plug-in* n1 de la fenêtre de montage, toutes ces variables sont remplacées par les arguments auxquels elles correspondent.

Dans le chapitre VII.4.5.1, nous avons vu que les données de date de départ, durée et adresse en Y étaient labélisées par un chiffre propre à chacun des ces différents types afin de pouvoir les identifier plus tard. C'est ici, dans ce patch que cette labélisation va être utilisée.

En bas à gauche de la copie d'écran de la page précédente, vous pouvez voir un objet **receive** #1 ce qui, dans notre exemple, équivaut à **receive** n1. Les données reçues par l'objet **receive** n1 pourront être identifiées grâce à l'objet **route** auquel il est connecté. L'objet **route** va aiguiller ce qu'il reçoit en entrée vers ses différentes sorties suivant le label qui leur a été affecté.

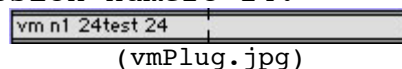
Voici, par exemple, le cheminement de la valeur 74 qui a été affectée à l'adresse en Y du *plug-in* ct dans la fenêtre de montage : on sait que l'adresse en Y du *plug-in* a été labélisée par le chiffre 4. Quand la liste 4 74 arrive dans l'objet **route**, le nombre 74 est immédiatement envoyé à l'objet **outlet** (sortie) connecté à la sortie de l'objet **route** correspondant à l'argument 4 de ce dernier. 74 continue son chemin, passe par la boîte nombre de l'interface utilisateur du *plug-in* afin d'en rafraîchir le contenu, puis est renvoyé dans le patcher de la copie d'écran par l'objet **inlet** qui se trouve en haut du patcher. Là, le nombre 74 est envoyé dans deux directions différentes :

- premièrement, il remplace la variable \$1 du message nsub #1 2 \$1 destiné à l'objet **coll**, ce qui donne le message suivant: nsub n1 2 74. Les données contenues dans l'objet **coll**, se présentent sous la forme suivante : n1, 1495.170166 86 ct 14503.855469 1; (voir le chapitre VII.4.3.1 pour l'ordre des données enregistrées en mémoire). n1, dans la syntaxe des données présentes en mémoire, joue le rôle d'indice. La donnée n1 du message nsub ci-dessus indique à l'objet **coll** que la suite de du message est destinée à la ligne n1. Le nombre 2 quant à lui, indique à l'objet **coll** que la suite du message est destinée à la valeur de rang deux après la virgule, et que celle-ci remplacera celle déjà présente. Le nombre 74 remplace ainsi le nombre 86 dans la ligne concernée. Le *plug-in* n1 est maintenant enregistré en mémoire comme ceci :  
n1, 1495.170166 74 ct 14503.855469 1
- deuxièmement, cette même valeur 74 est envoyée à l'objet **thispatcher** situé dans la fenêtre de montage, afin que la boîte objet correspondante au *plug-in* n1 s'y déplace à l'adresse 74 en Y

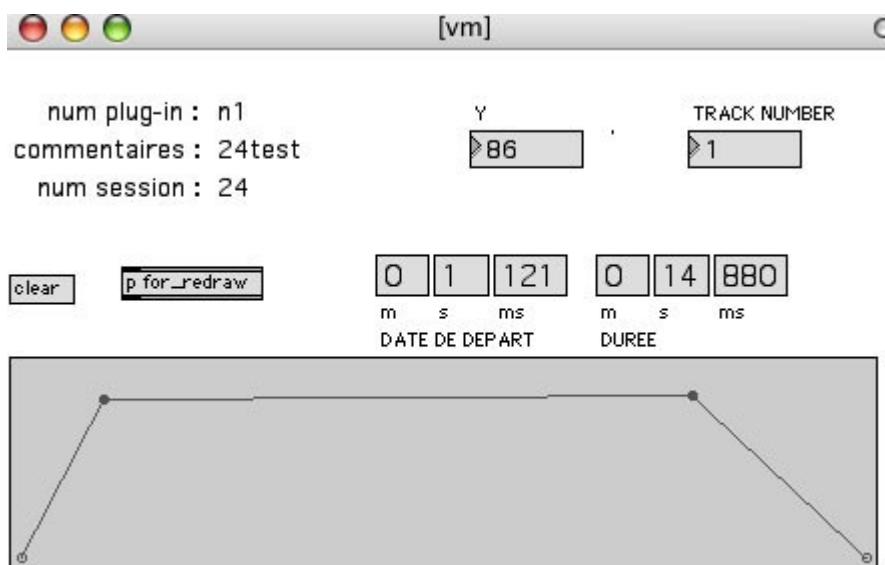
Ce type de procédé est valable pour toutes les autres données du *plug-in*.

#### VII.4.9.2- Description du *plug-in* pour le contrôle des sorties audio (*SoundLoudSpeaker*)

Voici l'exemple d'un *plug-in* de type sortie audio (*SoundLoudSpeaker*) et de nom n1, dont le commentaire est «24test» et qui appartient à la session numéro 24.



Comme nous avons tout-à-l'heure un *plug-in* ct pour le type contact, nous avons maintenant un *plug-in* vm pour volume (entendez par volume le niveau appliqué aux sorties audio). Une fois ce dernier créé dans la fenêtre de montage, voici l'interface qui se présente à l'utilisateur lorsque celui-ci double-clique dessus :



(vmPlugInterf.jpg)

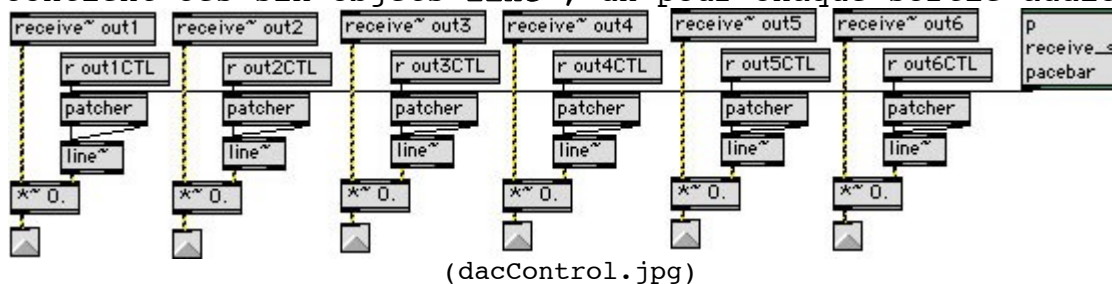
Bien entendu, comme pour les *plug-ins* de type contact, le *plug-in* vm est en fait une abstraction à trois arguments : nom du *plug-in* (n1), commentaire (24test) et enfin numéro de session (24). Tout ce qui concerne la gestion de la mémoire au sein de ce *plug-in* est exactement identique à ce que nous avons déjà vu pour le *plug-in* de type contact.

Concentrons-nous donc sur les particularités du type vm. La principale différence se trouve dans l'interface utilisateur. L'interface du *plug-in* n1 ci-dessus contient déjà des données mais lorsque ce dernier fut édité pour la première fois, il ne contenait aucune courbe et le paramètre *Track Number* était par défaut à 0 (c.a.d affecté à aucune sortie audio). Là encore, l'interface est d'une très grande simplicité d'utilisation. Les paramètres « DATE DE DÉPART » et « DURÉE » ont été définis par l'utilisateur en sélectionnant dans la fenêtre *Waveform* la portion de fichier audio auquel ce *plug-in* est affecté. La durée de la courbe est bien entendu égale à la durée du *plug-in*. La position en Y des points de la courbe est comprise dans un ambitus allant de 0 à 1. Ces valeurs correspondent à celles prévues dans l'environnement Max-MSP pour contrôler les niveaux de sortie audio (objet *dac~* = *Digital to Analog Converter*) .



L'objet utilisé pour tracer ces courbes de volume est l'objet **function**. Celui-même que nous utilisons déjà dans les versions antérieures. Dans cet objet, il est possible d'ajouter autant de points que l'on veut en cliquant à l'endroit désiré. Il est également possible d'effacer des points à l'aide de la combinaison « Ctrl-clic ». Le message `clear` permet d'effacer tous les points de la courbe d'un seul coup. Tous les paramètres ne sont envoyés qu'une fois l'interface du *plug-in* fermée. Le patcher `for_redraw` sert, quant à lui, au redessin de la courbe lorsque l'utilisateur double-clique sur le *plug-in* afin d'en modifier les paramètres.

Lors de la lecture de ce *plug-in* `vm`, les données correspondant aux différents points de la courbe seront envoyés à la fenêtre `dac` qui est accessible via le menu déroulant de la fenêtre `MMS`. Cette fenêtre `dac` est la fenêtre qui permet de mettre en marche ou d'arrêter le traitement audio. La fenêtre `dac` possède six vu-mètres, un par sortie audio, afin de permettre de contrôler les niveaux de sortie de l'application. Chacune de ces sorties sont contrôlées par un objet `line~` qui va se charger de procéder à l'interpolation entre les différents points de contrôle inscrits dans la courbe du *plug-in* `vm`. Comme nous l'avions vu dans les versions antérieures, l'objet **function** fut justement développé comme interface de contrôle pour l'objet `line~`. Voici le patcher qui contient ces six objets `line~`, un pour chaque sortie audio :



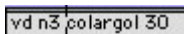
Le principe est semblable à celui utilisé pour la diffusion de musique acousmatique sur un acousmonium : le fichier audio est envoyé constamment sur toutes les sorties, et c'est le contrôle appliqué à chacune de ces différentes sorties qui va permettre de créer des mouvements du son dans l'espace, et donc de faire ce qu'on appelle de la spatialisation.

Le **patcher** `receive_spacebar` sur la droite n'a d'autre fonction que de couper toutes les sorties audio lorsque le bouton stop de la fenêtre *Waveform* est activé ou que l'utilisateur appuie sur la barre d'espace du clavier de son ordinateur pour stopper la lecture du fichier audio (un nouvel appui sur cette même barre d'espace relance la lecture et ainsi de suite...).

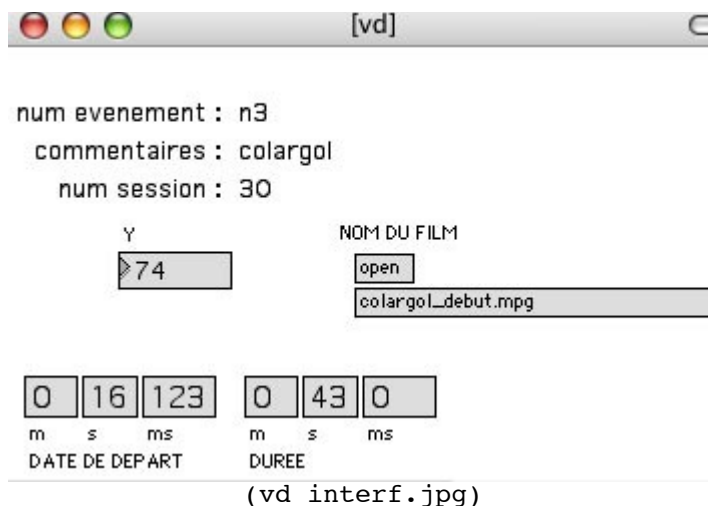


### VII.4.9.3- Description du *plug-in* vidéo

Voici l'exemple d'un *plug-in* de type vidéo et de nom n3, dont le commentaire est colargol et qui appartient à la session numéro 30.

  
(vd\_Plugin.jpg)

Il n'y a pas grand chose à en dire mais regardons en quand même rapidement l'interface utilisateur.



Comme vous l'avez certainement déjà remarqué, le type de *plug-in* vidéo est abrégé par vd. Quand l'utilisateur double-clique dessus afin d'en éditer le contenu, la fenêtre ci-dessus apparaît. On y retrouve les paramètres communs à chaque type de *plug-in* à savoir :

- l'adresse de positionnement en Y dans la fenêtre de montage
- la date de départ
- la durée

La seule différence dans le cas présent est que l'utilisateur va aller chercher, dans l'arborescence du disque dur de son ordinateur, le fichier vidéo qu'il désire faire jouer par ce *plug-in*. Pour cela, il suffit de cliquer sur le message open du paramètre « NOM DU FILM ». Une fenêtre de dialogue apparaît alors à l'écran. Celle-ci vous permet de vous déplacer dans l'arborescence du disque dur de votre ordinateur personnel et d'aller sélectionner le fichier vidéo que vous désirez utiliser. Une fois le fichier sélectionné, son nom apparaît sur l'interface du *plug-in* vd. Ici, dans notre exemple le nom du fichier est : colargol\_debut.mpg.

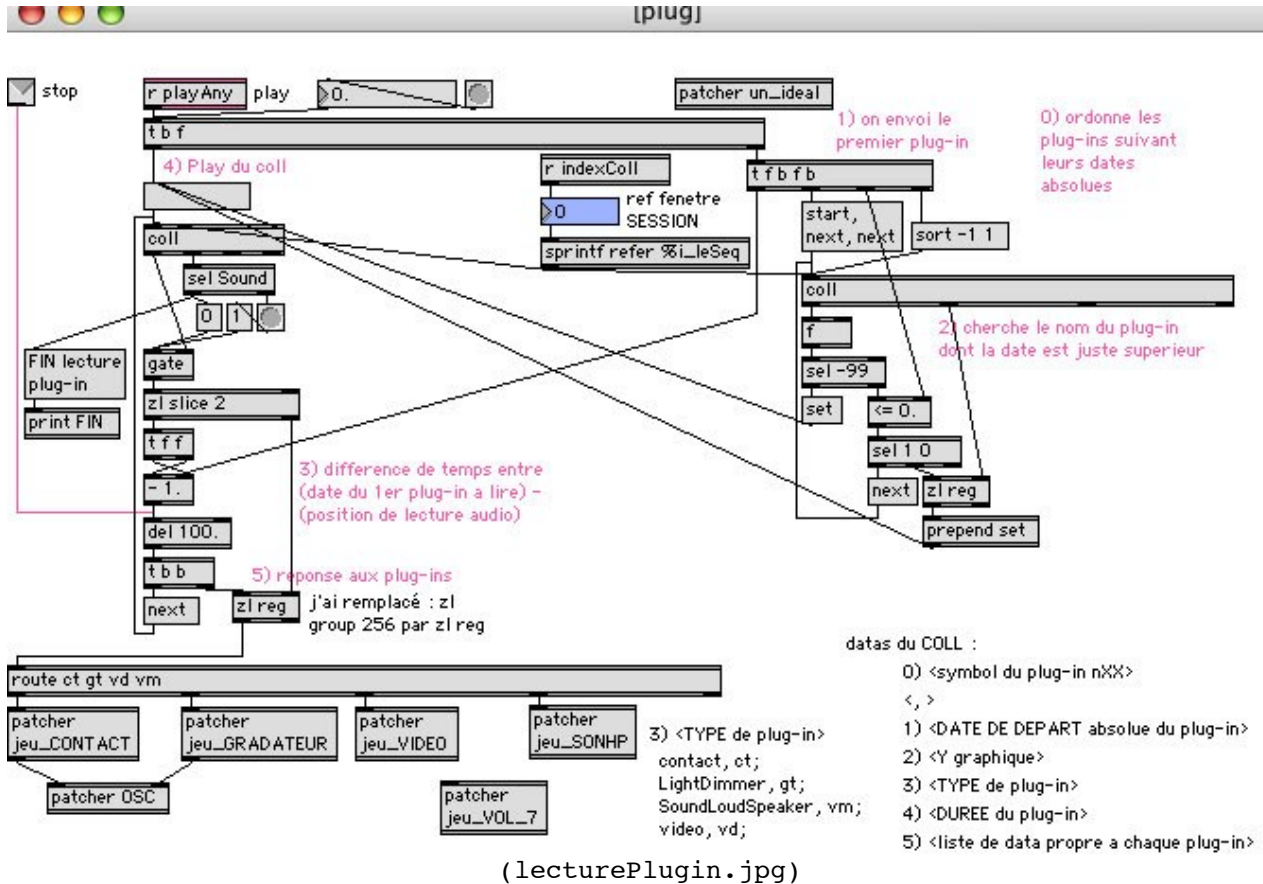
Comme tous les autres types de *plug-in*, les paramètres entrés ne seront validés qu'une fois la fenêtre du *plug-in* fermée.

Je ne décrirai pas le *plug-in* de type *lightDimmer* (gt pour gradateur) c'est à dire le *plug-in* qui sert au contrôle continu des lumières car il est la réplique exacte du *plug-in* de type *SoundLoudSpeaker* (vm).

Nous verrons dans le chapitre suivant comment les données propres à chaque type de *plug-in* sont lues et interprétées.

### VII.5- Lecture du montage

Nous en arrivons à la lecture du montage et donc à la lecture du contenu de la mémoire de la session. Analysons le patcher *plug* ci-dessous :



Dans MMS, ce patcher se trouve dans la fenêtre *Waveform*. Pour le visualiser, il faut que celle-ci soit en mode édition. La première donnée à entrer dans ce patcher est bien entendu le numéro de la session. Cette valeur est réceptionnée par l'objet **receive** **indexcoll** que vous connaissez bien désormais. Ce numéro est inséré dans un message **refer** et est envoyé aux différents objets **coll** se trouvant dans ce patcher afin que ceux-ci partagent la même mémoire. Vous connaissez très bien ce procédé maintenant.

Au moment de la lecture, c'est la position en millisecondes du pointeur de lecture qui entre dans notre patcher *plug* via l'objet **receive** **playAny**. Cette valeur est immédiatement envoyée dans un objet **trigger** qui va réagir de la façon suivante :

- la valeur qu'il reçoit de l'objet **r playAny** est envoyée par sa sortie droite. Cela a pour conséquence de séquencer une série d'évènements comme suit :
- les données contenues dans l'objet **coll** droite sont classées chronologiquement suivant la date de départ du *plug-in* grâce au message **sort -1 1** de manière à ce que le premier *plug-in* soit forcément situé en première ligne de la mémoire et que le

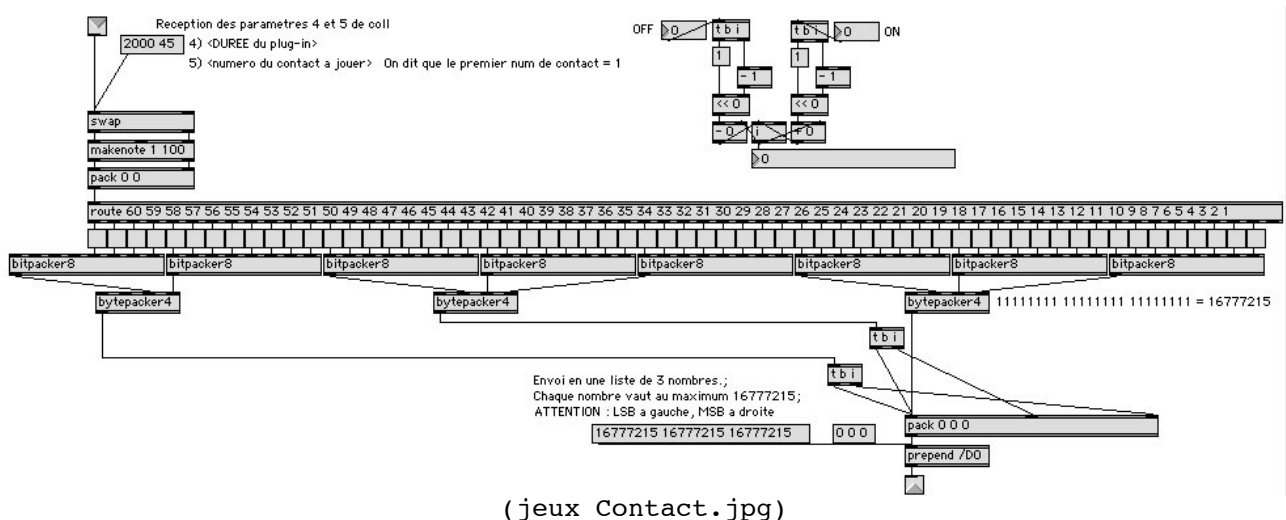
dernier soit en dernière ligne

- l'adresse du pointeur de lecture est comparée à l'adresse de départ du prochain *plug-in*. Tant qu'elle est inférieure, on sait que le *plug-in* suivant ne doit pas encore être activé. Si celle-ci est au moins égale à celle du prochain *plug-in*, cela signifie qu'il faut activer le *plug-in* suivant. Le but étant de toujours chercher le nom du *plug-in* à venir
- La sortie gauche envoie un message bang qui, lui, active la lecture du *plug-in* en cours
- à l'aide de l'objet **zl** slice 2, on ne garde que les données dont nous aurons besoin, à savoir le type de *plug-in* en cours et la liste des données qui lui sont affectées
- le tout est envoyé à un objet **route** qui va distribuer au type de *plug-in* correspondant, les données de jeux qu'il reçoit. Ainsi, les données de jeux propres au type contact ne seront envoyées qu'au patcher dont la fonction est de traiter les données propres à ce type de *plug-in* et ainsi de suite...

Au dessous de cet objet **route**, se trouvent ainsi quatre patchers différents, un pour chaque type de *plug-in*, chacun ayant en charge le traitement des données propres au type auquel ils sont affectés. Nous avons ainsi les patchers de noms :

- jeux\_CONTACT (pour le contrôle de lumières en mode tout ou rien)
- jeux\_GRADATEUR (pour le contrôle de lumières en mode continu)
- jeux\_VIDEO (pour jouer le fichier vidéo désiré)
- jeux\_SONHP (pour le contrôle des différentes sorties audio)

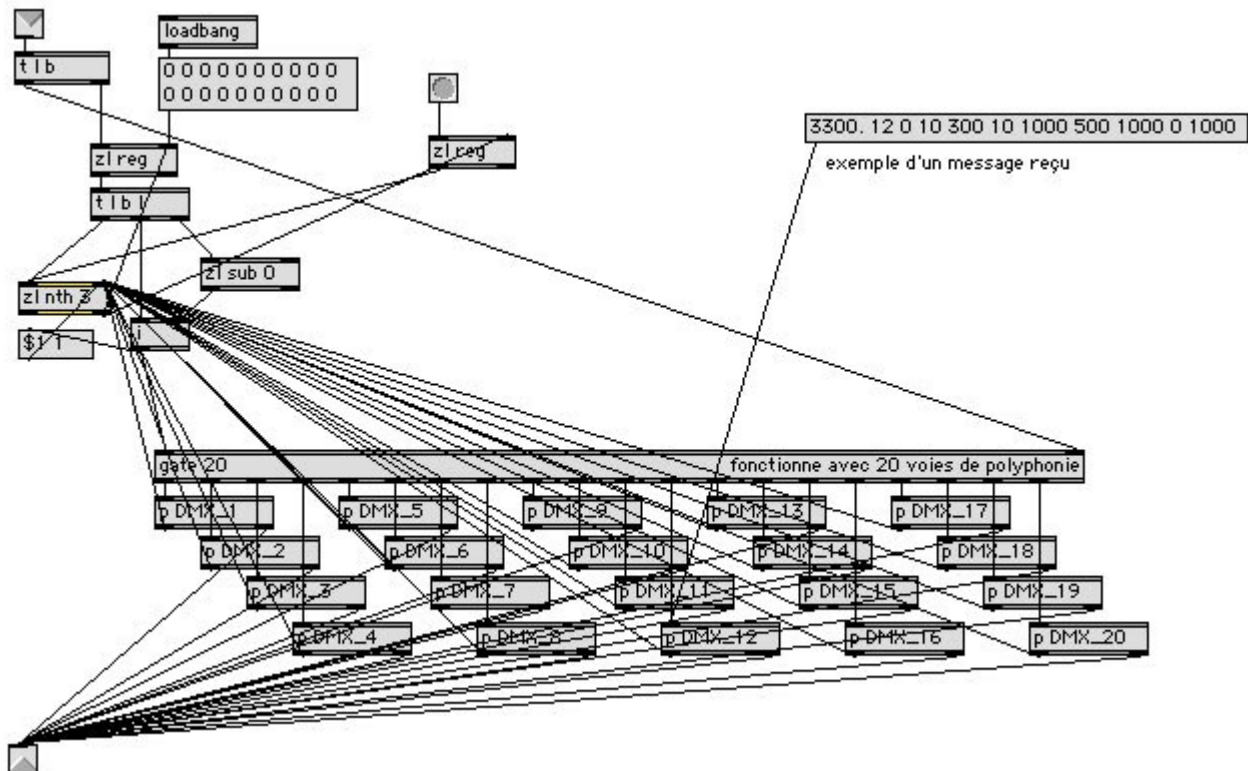
### VII.5.1- jeux\_CONTACT



Dans ce patcher arrivent seulement les éléments 4 et 5 de la mémoire, c'est à dire la durée du *plug-in* et le numéro de contact à jouer. Comme s'il s'agissait d'un événement MIDI, nous utilisons ici l'objet Max **makenote** qui va permettre de gérer la durée d'ouverture du contact en question en fonction de la durée du *plug-in*. Nous avons prévu la possibilité de gérer jusqu'à soixante contacts afin de prévoir une flexibilité logicielle mais la configuration des entrées/sorties de la carte GLUION telle que

nous l'avons commandée ne prévoit "que" trente et une sorties binaires ce qui permet quand même le contrôle de trente et un projecteurs en mode tout ou rien. L'information sortante est codée sous forme binaire. Ce message ainsi codé va spécifier à la carte la sortie binaire concernée ainsi que son état. L'état pouvant être de deux sortes uniquement : marche ou arrêt (tout ou rien). A chacune de ces sorties binaires est affecté un contact mécanique qui laissera passer ou non l'alimentation électrique au projecteur concerné.

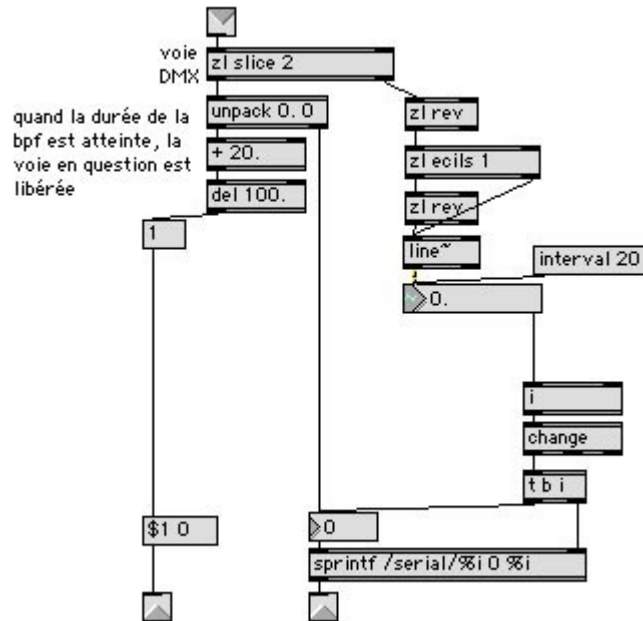
### VII.5.2- Jeux\_GRADATEUR



(jeux\_gradateur.jpg)

Dans ce patcher, ne pouvant pas bénéficier de l'objet **makenote** comme dans le patcher jeux\_CONTACT, nous avons dû gérer la possibilité de contrôler plusieurs projecteurs halogènes simultanément à l'aide des objets **zl nth**, **zl sub** et **zl reg**. Grâce à la combinaison de ces trois objets, il est possible de suivre l'état des différents canaux de traitement des données DMX de notre patcher et donc d'aiguiller les données reçues vers un canal libre via l'objet **gate**. Nous ne faisons finalement qu'utiliser une procédé analogue au système MIDI. On veut spécifier une certaine intensité lumineuse à un projecteur halogène donné. Ce projecteur est, bien entendu, affecté à une sortie DMX précise de la carte GLUION. Encore une fois, le but est de pouvoir contrôler plusieurs projecteurs indépendamment les uns des autres et en simultanée. C'est la raison pour laquelle nous avons prévu vingt voies de polyphonie possible ou, pour utiliser un terme commun au système MIDI, vingt canaux. Quand un canal est déjà occupé à envoyer des données de contrôle pour un projecteur sur une sortie DMX, nous allons tout de suite chercher quel est le premier canal disponible

pour le traitement des données de contrôle d'un second projecteur et de leur affectation à la bonne sortie DMX de la carte GLUION. Chacun des vingt canaux de traitement différents est contenu dans un patcher. Voici, par exemple, le patcher DMX\_1 qui contient le canal de traitement numéro 1 :



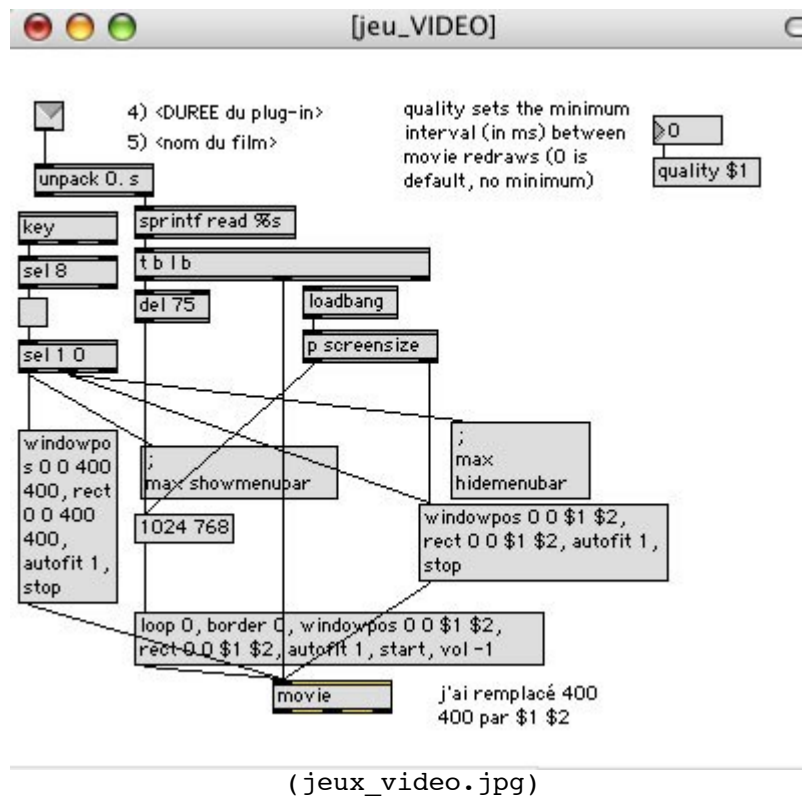
(DMX\_1.jpg)

Dans un patcher comme celui-ci, les données reçues arrivent sous forme de liste. Cette liste contient les informations suivantes :

- la durée du *plug-in*
- le numéro de la sortie DMX de la carte GLUION à laquelle les données de contrôle devront être envoyées
- la liste des données de contrôle

Cette liste est tout d'abord divisée en deux parties par l'objet **zl slice 2**. De sa sortie droite sortent les données de contrôle sous forme de liste. De sa sortie gauche sortent la durée du *plug-in* et le numéro de sortie DMX. Les données de contrôle sont envoyées un objet **line~** qui va les interpréter. Les valeurs fournies par **line~** sont ensuite acheminées dans l'entrée droite de l'objet **sprintf** et sont affectées à la deuxième variable **%i** du message que ce dernier doit assembler. La première variable de ce message est le numéro de sortie DMX. Le message ainsi constitué va communiquer à la carte GLUION que le message qu'elle reçoit est de type DMX et que la série de valeurs qui suivent sont destinées à sa sortie numéro N. La durée du *plug-in* est, quant à elle, affectée à un objet **delay** dont le rôle est de prévenir de la libération du canal une fois que la durée du *plug-in* est écoulé.

### VII.5.3- Jeux\_VIDEO



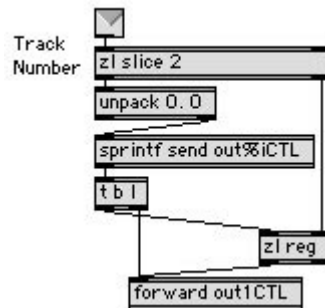
Ce patcher utilise l'objet **movie** pour jouer le fichier vidéo spécifié. Cet objet est en quelque sorte l'ancêtre de ce que sont aujourd'hui les objets **jit.qt.movie** et **jit.pwindow** dans Jitter mais avec beaucoup moins de fonctions possibles bien entendu. Au chargement du patch, tous les objets **loadbang** envoient un message bang par leur sortie gauche. Ici, le message bang est adressé à un patcher de nom **screensize**. Quand un message bang entre dans ce patcher, ce dernier envoie, par ses deux sorties, la taille de l'écran de l'ordinateur sur lequel est ouvert le patcher. Ces informations sont envoyées à divers messages afin que la fenêtre sur laquelle la vidéo sera jouée soit toujours de taille égale à celle de l'écran. Cette fenêtre est la fenêtre noire qui apparaît en arrière plan au sein de l'application MMS. Il est important que cette dernière occupe toujours la totalité de l'écran car c'est la copie de cette fenêtre qui sera projetée par le vidéo projecteur. L'utilisateur peut faire disparaître la barre de menus de Max en appuyant sur la touche **Back Tab** (Retour Arrière) de son clavier. Comme vous pouvez voir dans la copie d'écran ci-dessus, nous utilisons l'objet **key** en relation avec un objet **select 8 (sel 8)**. Le numéro 8 est en fait le code ASCII correspondant à cette touche **Back Tab**. Actionner cette touche provoque deux réactions différentes mais simultanées :

- la taille de la fenêtre vidéo change. Par défaut celle-ci occupe tout l'écran. En appuyant sur cette touche, l'utilisateur peut la ramener à une taille bien plus petite (400x400). Ré-appuyer dessus applique à nouveau la taille de l'écran à la fenêtre vidéo
- la barre de menus de Max disparaît ou apparaît suivant que la fenêtre vidéo est petite ou, au contraire, occupe le plein écran



Le nom du fichier vidéo à jouer est envoyé dans l'entrée gauche de l'objet **movie** et sa lecture en est immédiatement activée par le message `play`. Lors d'un spectacle utilisant la vidéo, il est bien entendu indispensable que cette fenêtre soit au premier plan.

#### VII.5.4- Jeux\_SONHP

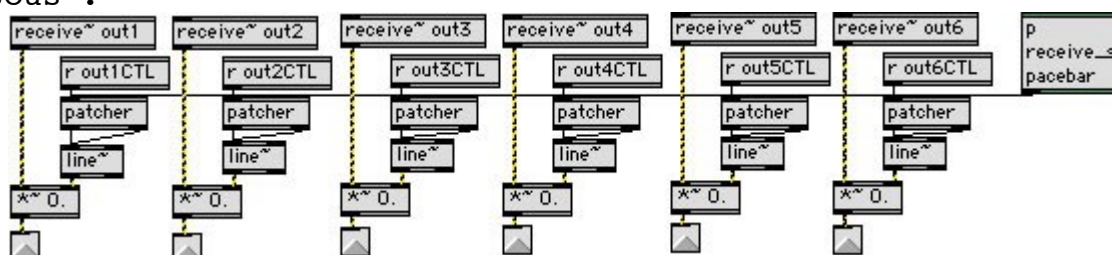


(jeux\_SONHP.jpg)

Dans ce patcher, les informations entrent sous forme d'une longue liste. Cette liste contient :

- le numéro de la sortie audio à contrôler
- les données de contrôle à lui appliquer

Ces deux parties de la liste sont séparées par l'objet **zl slice 2** situé à l'entrée du patcher. La liste des données de contrôle sont envoyées dans l'entrée droite de l'objet **zl reg** afin d'y être stockées et appelées ultérieurement. Le numéro de la sortie audio à contrôler est affecté à la variable `%i` du message `send out%iCTL` contenu dans l'objet **sprintf**. Ce message est ensuite envoyé à un objet **trigger**. Ce dernier le transmet à l'objet **forward** afin que celui-ci pointe désormais sur un objet **receive** ayant pour argument, par exemple, `out3CTL` (dans le cas où `%i` est égal à 3). L'objet **trigger** envoie ensuite un message `bang` à l'objet **zl reg** qui va ainsi transmettre le contenu de sa mémoire (la liste des données de contrôle) à l'objet **forward** qui, lui-même, le transmettra à l'objet **receive** sur lequel il pointe. MMS dispose de six sorties audio différentes, toutes regroupées dans la fenêtre `dac` et c'est à chacune de ces six sorties que sont rattachés ces différents objets **receive**. Les voici dans la copie d'écran ci-dessous :



(dacControl.jpg)

Nous avons déjà étudié ce patcher dans la partie consacrée au *plug-in* de contrôle des sorties audio au chapitre VII.4.9.2.

## **VIII- Création d'une nouvelle session MMS pas à pas**

Dans ce chapitre, je vais tout simplement décrire chacune des étapes à suivre pour créer une nouvelle session de travail dans MMS et y insérer un premier *plug-in*. Dans un souci de clarté, j'ai également réalisé, à l'aide d'un petit programme que j'ai développé avec Jitter, une saisie d'écran animée de chacune de ces différentes étapes. Vous pouvez trouver ce fichier vidéo sur le CD-Rom sous le nom `NewSessionMMS_Demo.mov`. Ce fichier vidéo est largement commenté et devrait être une aide visuelle précieuse en cas de doute. Ci-après, je décrirai exactement les mêmes étapes par écrit.

### **VIII.1- Créer une nouvelle session de travail**

- lancer l'application Max-MSP (version 4.6 au minimum)
- dans le menu *File* de Max-MSP, positionnez le curseur de votre souris sur *New*. Un sous-menu apparaît alors, cliquez sur `MMS_New_Session` et MMS ouvre alors une session vierge
- une série de fenêtres apparaissent à l'écran, cliquez sur la fenêtre située sur la gauche, celle où vous pouvez lire MMS, afin que cette dernière soit au premier plan
- dans cette fenêtre MMS, vous avez deux boîtes nombre (une rouge et une verte). Au dessus de la rouge figure le message `create new session`. Cliquez sur ce message, une fenêtre de saisie apparaît à l'écran
- saisissez-y un numéro (par exemple 54), fermez cette fenêtre et cliquez sur le numéro maintenant visible dans la boîte nombre rouge
- faites la même chose avec la boîte nombre verte en vous servant cette fois du message `open` situé au dessus de cette dernière
- votre nouvelle session de travail est maintenant créée et prête à être éditée. Cependant, si vous souhaitez la sauvegarder dès maintenant, reportez-vous dès à présent à la partie VIII.5

### **VIII.2- Charger un fichier audio dans la session**

- toujours dans la fenêtre MMS, vous trouverez un menu déroulant. Cliquez dessus et sélectionnez *dac*. Une fenêtre rouge apparaît
- dans cette fenêtre *dac*, vous trouverez un menu déroulant sur la gauche qui, par défaut, affiche `STOP`. Cliquez dessus et sélectionnez `START`. Cela permet d'activer l'audio de Max-MSP et donc par la même occasion l'audio de l'application MMS. La fenêtre *dac* change de couleur et devient verte afin de vous signaler que l'audio est bien en marche et que vous pouvez continuer
- rendez-vous maintenant dans la fenêtre *Waveform* afin de charger le fichier audio que vous désirez utiliser
- dans cette fenêtre, cliquez sur le message `open` et choisissez votre fichier grâce à la fenêtre de dialogue qui apparaît
- une fois votre fichier choisi, cliquez sur le message `play` de la fenêtre *Waveform* afin de charger le son dans la session
- attendez que le fichier audio soit entièrement chargé avant de faire quoi que ce soit



### VIII.3- Créer un *plug-in* dans la fenêtre de montage

- rendez-vous dans la fenêtre *Creation*
- cliquez sur le message *open*. Une fenêtre de saisie apparaît
- saisissez-y un commentaire pour votre *plug-in*. Ce commentaire doit être explicite car il vous servira ensuite de rappel sur la fonction de ce dernier (ex. *testSon*). Le commentaire ne doit contenir aucun espace ni caractère spécial comme par exemple : `^'?(>*$... etc.` Fermez la fenêtre de saisie
- dans le menu déroulant, sélectionnez le type *SoundLoudSpeaker*
- dans la boîte nombre insérez un numéro de votre choix. Par exemple, le numéro 1
- cliquez sur le bouton *create* afin que le *plug-in* soit créé dans la fenêtre de montage

Votre *plug-in* est maintenant visible dans la fenêtre de montage. Il devrait afficher des informations semblables à celles-ci suivant le commentaire et le numéro que vous avez donné :

```
vm n1 testSon 54
```

### VIII.4- Éditer le contenu d'un *plug-in*

- toujours dans la fenêtre *Waveform*, rendez-vous dans le menu déroulant qui doit afficher *Sound*, et sélectionnez-y le *plug-in* que vous souhaitez éditer. Dans notre exemple, il s'agit du *plug-in* n1
- toujours dans la fenêtre *Waveform*, sélectionnez une partie du fichier son et remarquez que la taille du *plug-in* que vous venez de créer est fonction de la longueur de votre sélection dans la fenêtre de montage. Double-cliquez sur le *plug-in* de même nom que celui que vous avez sélectionné dans le menu de la fenêtre *Waveform*. Le *plug-in* vous donne ainsi accès à ses différents paramètres
- dans le paramètre *Track Number*, entrez par exemple le numéro 1 afin de lui affecter le contrôle de niveau de la sortie audio numéro 1
- cliquez dans la partie grise de cette fenêtre de paramètres afin de créer votre courbe de contrôle qui sera appliquée à la sortie audio numéro 1. Pour créer un point, vous devez maintenir le bouton de la souris enfoncé au moment où vous cliquez. Déplacez votre point à l'endroit où vous le désirez et relâchez le bouton de votre souris. Pour effacer un point, faites « Ctrl-clic » sur le point en question. Comme il s'agit d'une courbe de volume, il est préférable que les premier et dernier points de la courbe de contrôle soient à zéro afin d'éviter tout clic audio. Le film de démonstration *NewSessionMMS\_Demo.mov* vous montre un exemple type d'une telle courbe
- fermez la fenêtre du *plug-in* afin que les différents paramètres que vous venez d'éditer puissent être pris en compte lors de la lecture de votre montage
- rendez-vous à nouveau dans la fenêtre *Waveform* et cliquez sur le message *play*. Quand le pointeur de lecture de la fenêtre de montage arrivera à votre *plug-in* de type *SoundLoudSpeaker*, son

contenu sera interprété et vous devriez avoir votre fichier audio affecté à la sortie 1 de l'application MMS. Pour vérifier que tout va bien, le VU-mètre de la sortie 1 dans la fenêtre dac doit afficher un signal de modulation audio

- pour arrêter la lecture, appuyez sur le bouton stop de la fenêtre *Waveform* ou encore sur la barre d'espace de votre clavier

#### **VIII.5- Sauvegarder la session pour la première fois**

- commencez par vous rendre dans le menu déroulant de la fenêtre dac et sélectionnez-y STOP afin de couper l'audio. Sauvegarder une session avec l'audio en marche n'est pas grave mais il est préférable de penser l'éteindre auparavant
- fermez la fenêtre dac à l'aide du bouton jaune commenté *close window*
- cliquez une fois sur la fenêtre MMS afin de la positionner au premier plan
- déplacez-vous dans le menu *File* de Max-MSP et sélectionnez *Save*
- sauvegardez toujours vos sessions dans le dossier My Sessions situé dans le dossier MMS\_All comme nous l'avons déjà vu
- nommez votre session en y ajoutant systématiquement l'extension .pat (ex. Sess54.pat)
- en bas de la fenêtre de sauvegarde, dans le champs Format, sélectionnez *Max Text File*
- cliquez maintenant sur *Save*
- votre session est désormais enregistrée sur votre disque dur, dans le dossier My\_sessions et sous le nom Sess54.pat

Pour fermer votre session, rendez-vous dans la fenêtre MMS et cliquez sur le bouton jaune *Close window* de cette dernière.

#### **VIII.6- Ouvrir une session existante**

Cette partie ne figure pas dans le film de démonstration. Voici comment procéder :

- rendez-vous dans le menu *File* de Max-MSP et cliquez sur *Open*
- allez chercher votre session dans le dossier My\_Sessions et double-cliquez dessus : elle s'ouvrira automatiquement
- cliquez une fois sur le numéro inscrit dans la boîte verte de la fenêtre MMS. Vous ne pourrez rien faire tant que vous n'aurez pas fait cela
- chargez à nouveau votre fichier audio qui, lui, n'est pas sauvegardé avec la session
- une fois le fichier audio entièrement chargé, vous êtes prêt

## IX- Perspectives d'évolutions pour l'application MMS

### IX.1- La vidéo

L'application MMS, telle qu'elle est actuellement développée, n'utilise pas la troisième couche de Max pour la vidéo, c'est à dire la partie Jitter. La raison pour laquelle nous avons fait ce choix est essentiellement une question budgétaire pour Monsieur BERARD. Sur le site internet de Cycling74, Max-MSP coûte \$495, la partie Jitter est en supplément et coûte à elle seule \$395. Il est cependant possible d'acheter le tout comme un ensemble et dans ce cas le prix total s'élève à \$850 ce qui est relativement onéreux. Aussi, comme la partie logicielle dédiée à la vidéo se limite à de la lecture sans le moindre traitement, il aurait été dommage de rendre indispensable l'acquisition de Jitter alors que nous n'en aurions utilisé que deux objets tout au plus (**jit.qt.movie** et **jit.pwindow**). C'est pourquoi nous avons pris la décision d'utiliser l'objet Max **movie** qui constitue les prémices de la troisième couche de Max maintenant entièrement dédiée à la vidéo : Jitter. L'avantage financier est important et c'est ce qui fait que notre choix se défend.

Cependant, il est vrai que Jitter permettrait d'apporter de nombreuses améliorations ou optimisations non négligeables à l'application MMS. L'image pourrait être de meilleure qualité car Jitter offre des outils qui permettent d'effectuer de nombreuses modifications comme par exemple le contraste, le lissage de l'image si celle-ci est trop pixélisée, un réglage indépendant des quatre composantes ARGB de l'image (Alpha, Red, Green et Blue). Il est également possible d'appliquer à l'image une multitude d'effets pré-existants (les effets *Quick Time*) mais aussi de construire ses propres effets. On pourrait imaginer des interactions son/images très intéressantes et de surcroît fortement utilisées de nos jours dans les spectacles multimédia... etc. Bref, les possibilités offertes par Jitter sont considérables et extrêmement puissantes, surtout qu'elles peuvent être utilisées en temps-réel. Il est vrai que Jitter est relativement onéreux mais sa puissance le justifie entièrement. En utilisant Jitter pour la programmation de MMS, il est possible d'envisager une infinité de fonctions supplémentaires qui ne sont tout simplement pas possibles autrement. Pour ces raisons, le *plug-in* vidéo sera certainement amené à être revu entièrement dans les versions futures, et sa programmation bénéficiera de la puissance et de la flexibilité offertes par Jitter.

### IX.2- Le son

Je ne pense pas intégrer de possibilité de traitement audio au sein de l'application MMS, car celle-ci se veut un séquenceur permettant de synchroniser des événements de types vidéo, son et lumière pour des spectacles multimédia. Cependant, pour des raisons d'acoustique, il peut être intéressant d'avoir une possibilité d'ajouter de la réverbération afin que le son puisse être intégré au mieux à l'acoustique du lieu dans lequel le spectacle sera présenté.

### **IX.3- L'interface utilisateur**

Bien que cette interface soit déjà très performante, il y a certainement des points qui seront amenés à être améliorés à la suite de commentaires des différents utilisateurs, car mon propre point de vue n'est plus assez objectif. L'esthétique sera certainement le premier point à changer. Je pense lui donner un aspect aluminium comme la plupart des applications fonctionnant sous Mac OS X. Tout ceci peut passer pour un simple détail, mais le *design* de l'interface a beaucoup d'importance pour la personne qui est amenée à travailler très régulièrement avec l'application. Une interface accueillante fera toute la différence en terme d'incitation.

### **IX.4- Au-delà du CNSMD**

Depuis déjà quelques mois, j'ai dans l'idée de transformer MMS en projet Libre accessible à tous. Tous les codes source, le logiciel, une documentation utilisateur ainsi qu'une documentation développeurs seront disponibles sur un site *Web*. Je compte également mettre en place un forum de discussions instantanées (*chat*) afin que les différents développeurs qui voudront participer au développement de MMS puissent discuter entre eux, échanger leurs idées et partager leurs problèmes de développement en temps-réel. Une *mailing list* pourra également être mise en service pour les différents utilisateurs et testeurs. Il existe déjà de nombreux projets organisés de la sorte sur le site internet : cf. <http://www.sourceforge.net> qui est la référence dans le domaine de l'hébergement de projets de logiciels libres. Si le projet est clairement présenté, il trouvera rapidement d'autres développeurs qui voudront y participer et le faire grandir. Plus le projet mobilisera de développeurs et plus l'application sera performante, fiable, reconnue et utilisée des professionnels du spectacle, que ce soit dans le domaine de la composition, de la chorégraphie, des musiques actuelles ou encore du théâtre. C'est ainsi que l'application MMS n'en est finalement qu'à son début...

Je conseille à ce sujet, l'ouvrage de Monsieur Karl FOGEL cité dans les références.

## Références

Voici quelques références qui m'ont été utiles pour la programmation de MMS ainsi que pour la rédaction de ce mémoire.

### Livres :

- Karl FOGEL, « *Producing Open Source Software* » (*How To Run a Successful Free Software Project*), O'REILLY (ISBN 0-596-00759-0)
- Claude DELANNOY, « Best Of » (Langage C), EYROLLES (ISBN 2-212-11123-1)
- Douglas COMER, « *Internetworking With TCP/IP* » (*Principles, Protocols, and Architecture*), PRENTICE HALL (ISBN 0-13-470154-2)
- Nicolas COLLINS, « *Handmade Electronic Music* » (*The Art Of Hardware Hacking*), ROUTLEDGE (ISBN 0-415-97592-1)
- Forrest M., « *Getting Started in Electronics* », Mims III
- Tom Igoe et Dan O'Sullivan, « *Physical Computing: Sensing and Controlling the Physical World with Computers* », THOMSON COURSE TECHNOLOGY

### Liens internet :

#### **Renseignements généraux :**

- Wikipedia, l'encyclopédie libre du Net : <http://www.wikipedia.org>
- MS-DOS : <http://fr.wikipedia.org/wiki/MS-DOS>
- MOELLER : <http://www.moeller.fr>
- CNMAT Berkeley : <http://www.cnmat.berkeley.edu/>  
OpenSoundControl : <http://www.opensoundcontrol.org/cnmat>
- Cycling '74 : <http://www.cycling74.com>

#### **UDP-TCP/IP :**

- suite des protocoles : [http://fr.wikipedia.org/wiki/Suite\\_des\\_protocoles\\_internet](http://fr.wikipedia.org/wiki/Suite_des_protocoles_internet)

#### **Ethernet :**

<http://fr.wikipedia.org/wiki/Ethernet>

**Oscqoop** : (oscilloscope distribué sous licence libre)

<http://lsn.unige.ch/osqoop/index-fr.html>

#### **Source Forge :**

<http://www.sourceforge.net>

#### **Licences GNU :**

<http://www.gnu.org/licenses/>

#### **Forum Neues Musiktheater :**

<http://www.fnm.de>

[http://www.fnm.de/index\\_frame.php?lan=en&n=3&s=0](http://www.fnm.de/index_frame.php?lan=en&n=3&s=0)

**Association LIEU et projet synArt** : <http://www.synart.org>

**Carte GLUION** : <http://www.glui.de/prod/gluion.html>

**Vignéroscope :**

<http://perso.orange.fr/pascal.blouzard/rubriques/region/vigneroscope/vigneroscope.htm>

**interface-z :**

<http://www.interface-z.com/>

Je conseille particulièrement le site d'interface-z car on y trouve non seulement du matériel extrêmement bien adapté au milieu du spectacle contemporain mais également car ils proposent une série de documentations particulièrement bien faites sur les protocoles de communications, comment souder et assembler des capteurs, qu'est-ce qu'un gradateur... etc.

### **Remerciements**

- **Denis LORRAIN** pour ses relectures et corrections, ses conseils ainsi que sa patience
- **François ROUX** pour son aide précieuse au développement informatique et électronique
- **Jean HOLLEVILLE** pour les cours particuliers en langage C et Unix
- **Emmanuel JOURDAN** pour ses conseils en développement javascript
- **Sukandar KARTADINATA** pour ses conseils techniques
- **Philippe BERARD** pour ses participations diverses
- **Carl FAÏA** pour son soutien et sa confiance depuis maintenant plusieurs années
- **la SACEM** pour son aide financière tout au long de mes études supérieures au CNSMD de Lyon
- **Sylvie FAZARI** pour son soutien sans lequel mes études au CNSMD de Lyon n'auraient tout simplement pas été possibles

# **Annexes**





## Quelques mots sur mon expérience d'assistant musical au FNM de Stuttgart

En parallèle à mes études, avant même mon entrée au CNSMD de Lyon, j'ai toujours cherché à participer à des projets extra-scolaires afin d'être le plus en phase possible avec une réalité de terrain. En 2006, de mi-mai à fin juillet, j'ai eu l'occasion de travailler comme assistant musical au *Forum Neues Musiktheater* (FNM) de Stuttgart en Allemagne, à la réalisation d'une pièce de Alan HILARIO, intitulée "*Schoner Gotterfunken ! bilder einer ausstellung einer ausstellung*", d'une durée de cinquante quatre minutes pour deux saxophones, percussions, traitements audio et interactions réseaux en temps-réel ainsi que projection vidéo. Cette expérience fut des plus enrichissante tant au niveau des connaissances que j'y ai acquises, de l'expérience d'une méthode de travail, de la gestion des divers problèmes d'une création, mais aussi de l'acquisition d'une certaine confiance en moi que je ne possédais pas auparavant.

La grande différence entre le FNM et tout autre centre de recherche musicale, est son orientation artistique. En effet, le FNM s'est spécialisé dans le théâtre musical. Le domaine du théâtre musical est bien différent de celui de la création musicale pure. Comme vous n'êtes pas sans savoir, le théâtre a ses exigences et ses besoins de flexibilité immédiates que la musique électroacoustique ne demande pas forcément... du moins pas de la même façon. En milieu théâtral, l'assistant est amené à travailler avec le compositeur, certes, mais également avec des musiciens, des comédiens, un metteur en scène, les éclairagistes et bien entendu l'ingénieur du son. Toute cette équipe de spécialistes se doit de cohabiter au mieux dans des délais extrêmement courts, ce qui n'est pas toujours évident.

L'exemple le plus frappant qui me vienne, afin de traduire ce besoin de souplesse immédiate, concerne la collaboration avec le metteur en scène. En milieu théâtral, il est d'usage de prendre des décisions importantes pour la mise en scène lors des répétitions. Il est relativement facile de tester différents mouvements, différents costumes ou encore différents textes avec un comédien, mais il est beaucoup moins évident de modifier un programme informatique en moins de cinq minutes. Aussi avisé des nouvelles technologies que puisse être le metteur en scène, il lui sera toujours relativement difficile d'admettre ce fait. Pour lui, la réplique suivante est pratiquement impensable «Oui, je peux faire ça mais donne-moi vingt minutes». Vingt minutes, c'est trop long, car il faut pouvoir tester rapidement afin de pouvoir avancer. Bien entendu, vingt minutes au cours d'une répétition d'orchestre, c'est également impensable mais pour d'autres raisons : l'orchestre travaille généralement à heures fixes. Ses horaires sont comme gravées dans le marbre.

Au théâtre, le problème vient du fait que le spectacle se construit durant les répétitions, et qu'il faut donc pouvoir tester un maximum de choses en un minimum de temps, afin de pouvoir prendre des décisions et que tout le reste de l'équipe puisse effectuer les réglages nécessaires en fonction de ces décisions (la régie lumières particulièrement). Aussi, malgré le

fait que la musique soit pensée pour instruments et technologies, étrangement, la technologie passera toujours au dernier plan. Comme si, de toute façon, tout était possible avec l'ordinateur. Pourtant, toute personne ayant été amenée à travailler avec les technologies, sait que cela demande beaucoup d'essais, de paramétrages et de vérifications diverses.

Comme je le disais plus tôt, un projet de théâtre musical rassemble diverses spécialités. Chacune d'elles mériterait un temps de préparation propre qui ne lui est que rarement accordé. C'est un paramètre important dont chacun doit s'accommoder tout en arrivant finalement à trouver le temps minimal nécessaire pour assurer la qualité de sa contribution et l'interaction avec les autres spécialités. Dans le même temps, je n'avais jamais vu une production bénéficier d'autant de possibilités de répétition. Deux mois et demi constituent une grande chance accordée au compositeur invité en résidence pour une telle production. En temps normal, si le projet est relativement bien pensé dès le début des répétitions (ce qui ne semble jamais être le cas) le temps mis à disposition devrait laisser suffisamment de latitude pour la réalisation et une certaine dose d'expérimentation.

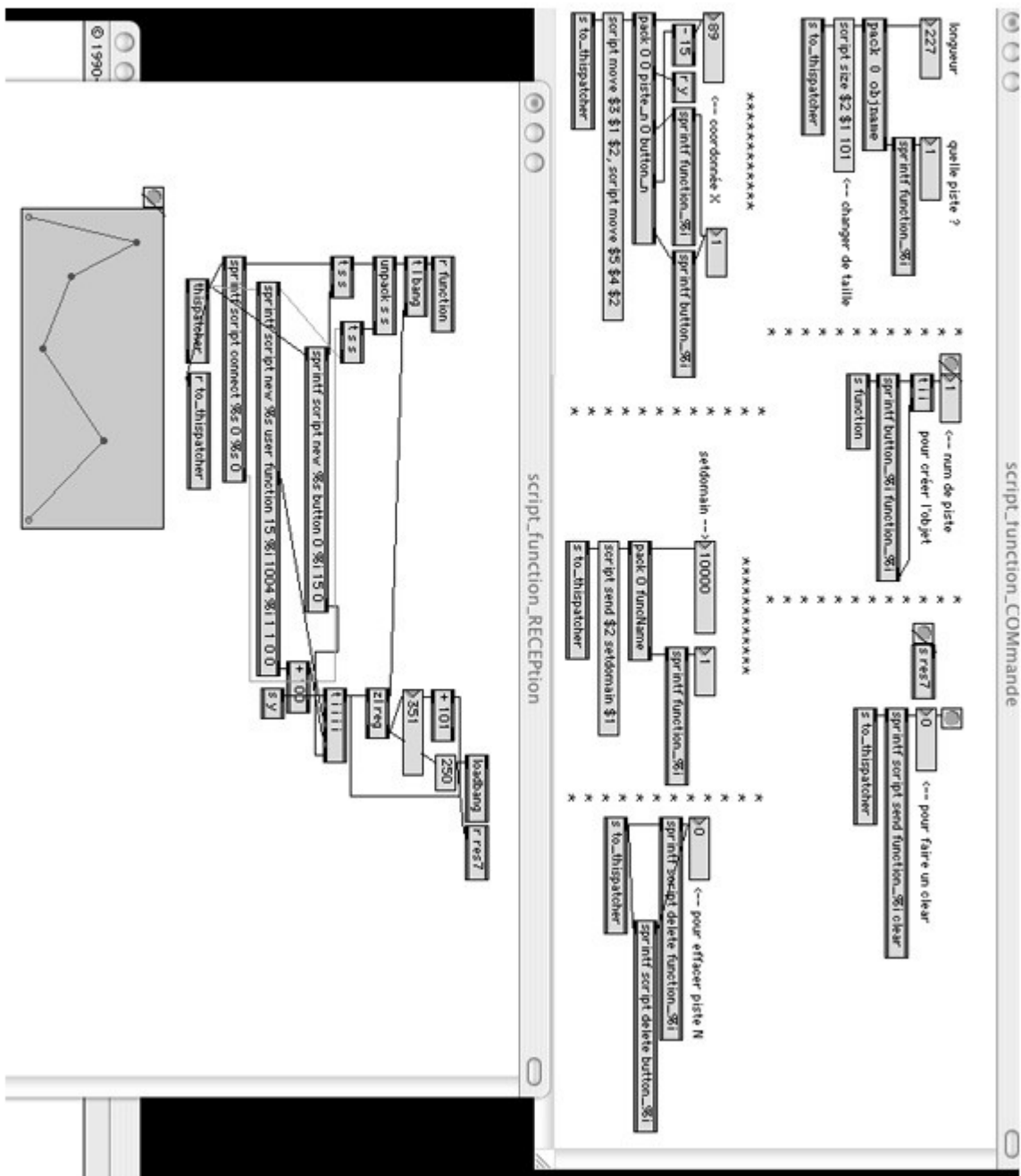
Ce projet au FNM m'aura beaucoup appris. Savoir dire non tout en faisant comprendre au compositeur qu'on ne le fait pas par plaisir n'est pas chose facile, mais malheureusement parfois indispensable. J'ai appris à programmer plus rapidement que je n'aurais pensé pouvoir le faire. Enfin, j'ai également beaucoup appris sur la communication au sein d'une équipe. La somme des difficultés auxquelles j'ai eu à faire face, ainsi que le stress constant dû au travail toujours réalisé dans l'urgence m'ont finalement montré que j'étais capable de gérer cet ensemble de contraintes et mener le projet à son terme.

Le 28 juillet 2006, la première se tenait au FNM de Stuttgart, et mes trois ordinateurs reliés travaillant en réseau ont passé l'épreuve sans le moindre problème... à mon grand soulagement !

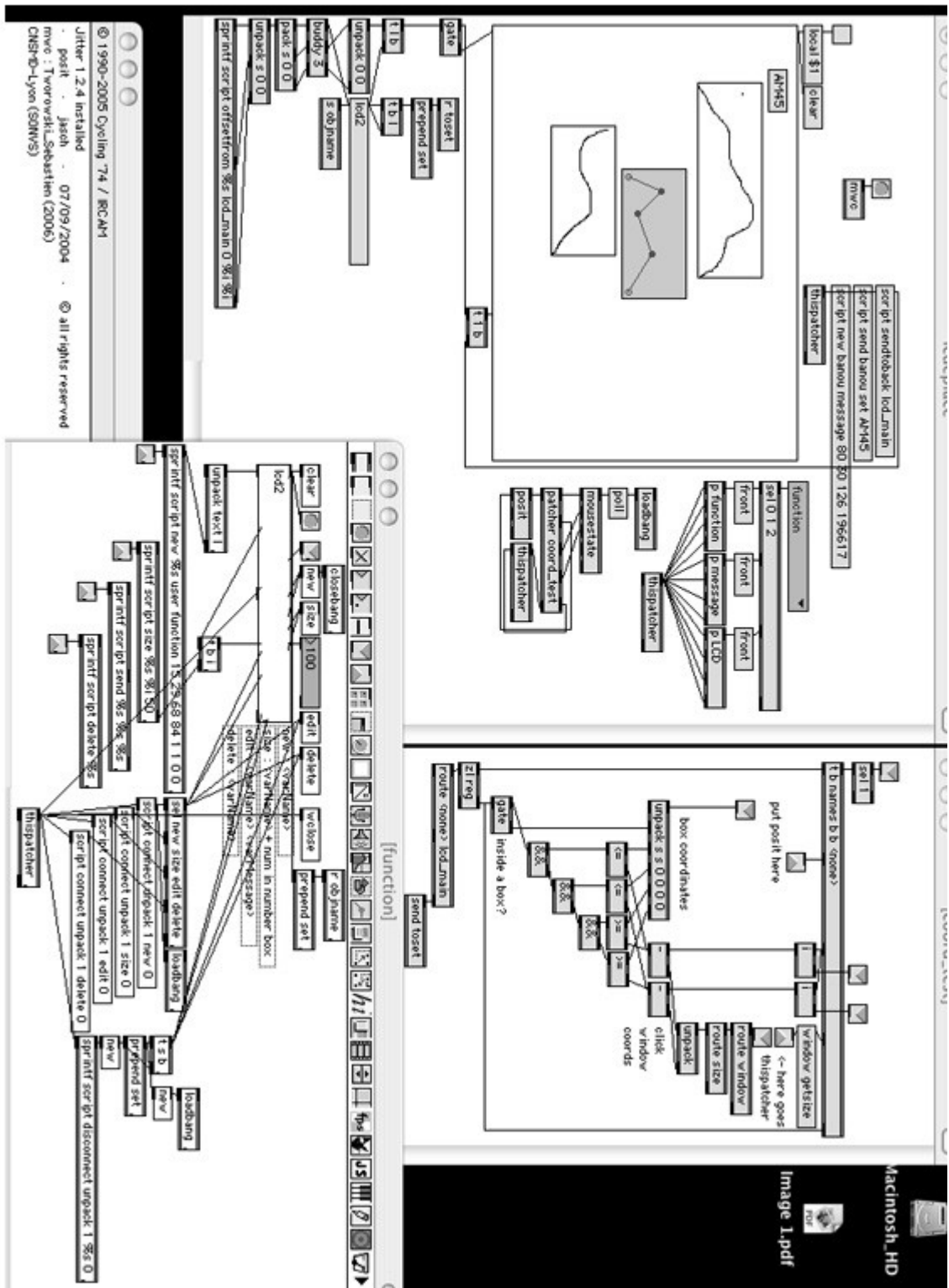
# Photos



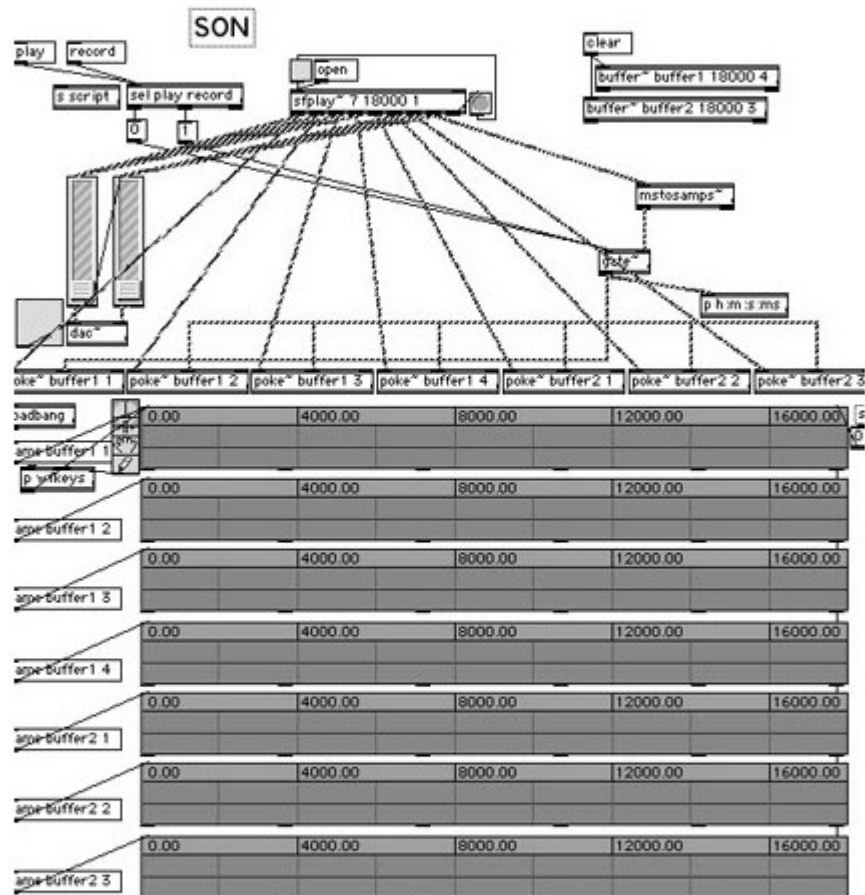
## 1- ScriptComAndRec



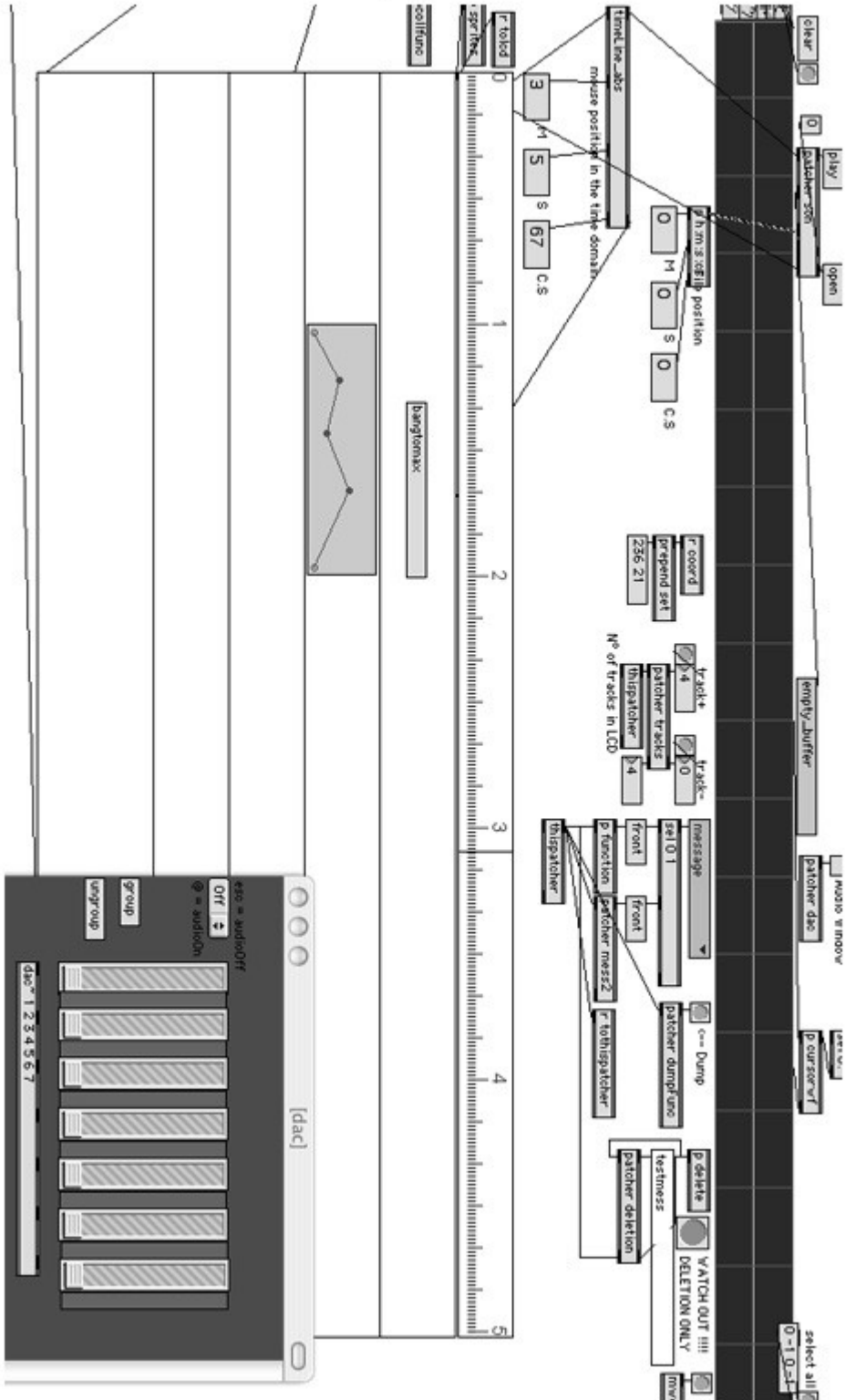
## 2- Lcdeplace



### 3- multiCanal

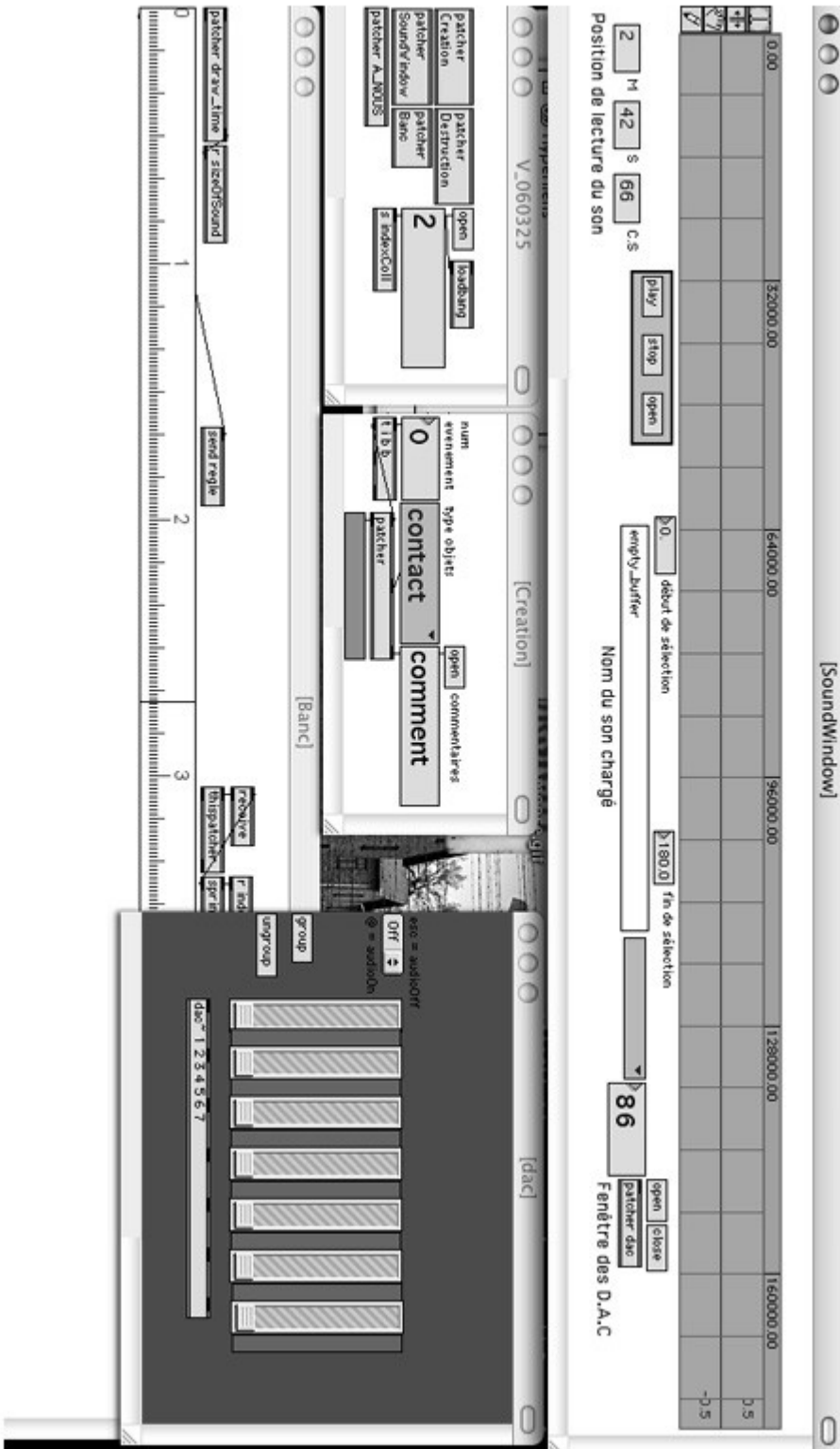


#### 4- interf\_proviNew

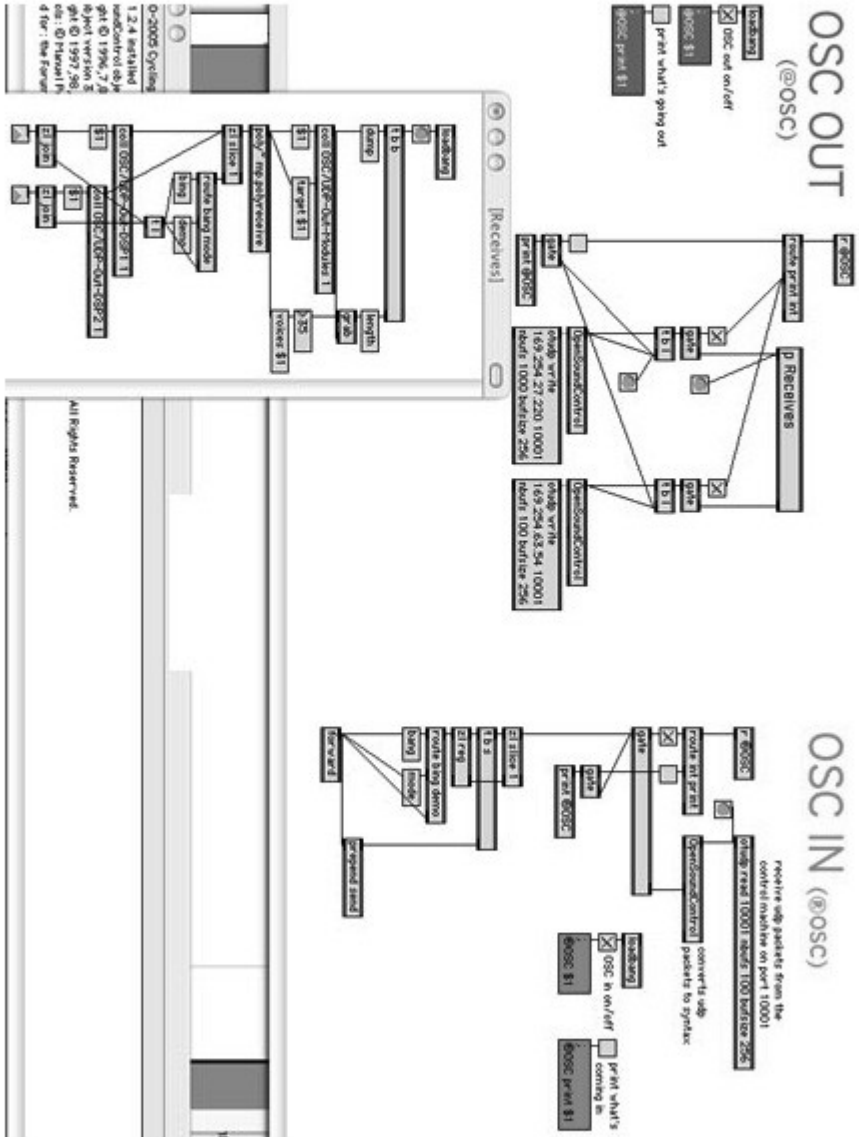




**5- demo060325**



## 6- OSC\_basics



## 7- règle\_interne

