Rapport de Soutenance n^{o}2

2B2S

Thomas 'billich' De Grivel
Maxime 'loucha_m' Louchart
Bruno 'Broen' Malaquin
Julien 'Splin' Valentin

Patchwork13!

Table des matières

Chapitre 1

Introduction

Nous voici à la seconde soutenance du projet Patchwork13, l'outil de synthése modulaire !

Entre ces deux soutenance, le planning a été respecté, tant au niveau du cluster qu'au niveau de l'interface, nous sommes même en avance sur les deux. De plus des rajouts ont été réalisés dans le noyau permettant une gestion plus large des données, ainsi qu'une exportation en xml.

Sur le cluster, basés sur la base réalisée lors de la première soutenance, nous avons implementé la quasi integralité des messages de commande du server.

Concernant l'interface graphique, une fonction de drag and drop à été concue pour permettre un affichage complet du patchwork que l'on veut créer. De plus dans ce graphe les éléments sont réellement reliés, ce n'est pas qu'un simple affichage.

Au niveau de la SDL, des patchs de son et dáffichage en openGL ont été créés afin d'avoir des patchs visuels et auditifs utilisables et qui donneront une possibilité de présenter quelque chose de plutôt impressionant pour les démonstrations qui viendront aux futures soutenances.

Sur ce, passons tout de suite à la presentation plus approfondie de notre travail.

Chapitre 2

Noyau

2.1 Gestion des données

La gestion des données a été complètement changée pour pouvoir connaitre la taille des données et gérer les tableaux. On distinguera désormais un bloc de données d'un segment de données.

Un bloc de données est en fait une union de la taille d'un int (32 bits) où l'on peut donc mettre 4 `char`, 2 `short` ou 1 `long`, signés ou non. L'union incluait précédemment un `long long` qui amenait la taile à 64 bits, taille moins adaptée aux processeurs actuels. Nous avons donc supprimé ce champ, puisqu'il est possible de gérer des segments de plus d'un bloc.

Un segment de données est un tableau de blocs. Le premier bloc indique le nombre de blocs suivants, donc si le nombre de blocs est zéro le segment comporte un seul bloc, si le nombre de blocs est 1, le segment contient 2 blocs au total.

Cela nous permet de gérer des tableaux ainsi que des types structurés plus complexes. Auparavant il était possible de metre un pointeur dans l'union mais du coup il était impossible de connaitre la taille des données pointées.

Il était nécessaire de connaitre la taille des données pour le cluster, car celui-ci devra transferer des données d'un ordinateur à l'autre. Cela pourra également servir à sérialiser les données dans un flux et à les lire, sans avoir a faire une fonction pour chaque type de données.

Nous avons donc du mettre à jour la librairie des patchs standard pour qu'elle utilise les nouveaux types de donées, c'est à dire récrire toutes les macros pour chaque type (float, int, *etc.*), et adapter chaque patch pour qu'ils utilisent correctement les nouvelles macros.

Cette nouvelle gestion des données a donc demandé du travail, mais donne beaucoup plus de puissance et de possibilités puisque l'on connait et qu'on peut faire varier leur taille, y mettre un tableau ou même une structure.
Cette nouvelle méthode ressemble beaucoup à la gestion des variables dans Objective CAML, référence sûre.

2.2 Exportation en XML

Il est désormais possible d'exporter un patchwork dans un fichier XML, cela grace à la librairie `libxml2`. Une fonction d'exportation est disponible pour chaque objet du noyau : patchwork, patch_class, patch, input, output, data_type, data...

Le lien avec l'interface graphique est fait : la commande "sauvegarder" dans le menu de la fenetre d'un patchwork permet de sélectionner un nom de fichier et exporte le patchwork en XML.

L'importation à partir d'un XML est prévue pour la prochaine soutenance. Ces deux fonctionnalités n'étaient pas prévues dans le cahier des charges mais nous avons trouvé qu'elles étaient un élément important pour pouvoir tester et utliser le projet.

Chapitre 3

Librairie de patches standards

3.1 Données

Tous les types de la librairie standard ont été changés pour s'adapter à la nouvelle gestion des données, qui permet désormais de gérer la taille des données et les tableaux (voir 2.1 **Gestion des données**, page 3).

3.2 Type vecteur

3.3 Exemple de nouveau type : Vect4f

À la base nous avons créé ce type pour l'OpenGL, afin de contenir les coordonnées d'un vecteur pour la 3D. Grâce au type de base déjà existant et en les adaptant on peut facilement en inventer.

Par exemple pour vect4f on créé un dossier avec un petit header dedans contenant des `#define` pour définir les nouveaux input et output mais aussi spécialement ici une fonction lui permettant de faire une variable facilement tout en la remplissant. Bien sûr ensuite une multitude de patches accompagne ce type pour lui faire faire les fonctions courrantes telles que addition, soustraction, constante, affichage.

3.3.1 le type

Pour ce type on voulait avoir un tableau statique de 4 cases mémoires. Pour que se fasse, on utilise une fonction se trouvant dans le noyau du projet qui initialise un tableau (de 4 cases, logique) et on le remplis avec des élément génériques pris en paramêtres que l'on caste avec l'union contenant tous les types très simples.

3.3.2 les patchs dépendants

Les patchs sont gérés comme les autres avec la fonction “pump” et “init”. Le type étant un peu plus complexe pour cet exemple, il a fallu ajouter des “defines” au types pour aisément manipuler les données de chaque case mémoire dans les inputs et outputs.

3.4 Extensions possibles

Il y a une infinité de patches à écrire, surtout que l'on peut utiliser des types plus complexes. L'objet de notre projet est pourtant plus leur mise en route et leur moyens de fonctionnement car nous pensons que cela encouragera plus de gens à utiliser la librairie et donc à écrire d'autres patches.

Nous écrirons donc surement quelques patches pour illustrer le fonctionnement du noyau, du cluster, de l'interface graphique et des librairies que nous écrivons (SDL OpenGL et son), mais la librairie standard n'aura plus de travail officiellement prévu pour cete année.

Chapitre 4

Cluster

4.1 Présentation générale et définition du cluster

Un cluster est un groupe de serveurs indépendants fonctionnant comme un seul et même système.

Un client dialogue avec un cluster comme s'il s'agissait d'une machine unique. Ce systême permet donc d'augmenter considérablement les performances lors du traitement désiré.

4.2 Rappel du fonctionnement général

Le cluster est divisé en deux parties :

Un client, lancé sur la machine sur laquelle on veut utiliser patchwork13, et un nombre indéfini de serveurs qui sont là pour faire les tâches de traitement demandées par le client.

La communication entre le client et les servers s'effectue en deux temps :
- une identification en UDP (broadcast)
- Connection et dialogue entre client et server en TCP

4.3 Ce qui a été fait

4.3.1 Définition des types utilisés

Pour avoir un cluster simple mais complet, nous avons définis différents types qui sont en fait des structs :

- Le type server contient un descripteur de la socket qui est en écoute permannente de nouvelle connection TCP, ainsi qu'une liste de structures client.

- Le type client (client du server) contient un descripteur de socket sur laquelle le client (là où est lancé patchwork13, où l'output d'un patch) est connecté, un pointeur sur le patchwork qui tourne sur le server, un pointeur sur le thread correspondant aux communications avec le client connecté., une liste d'outputs (outputs reseau), une liste d'inputs (inputs reseau) et un pointeur sur le server auquel il appartient.

- Le type input réseau contient un pointeur sur input, un descripteur de socket pour savoir avec qui cette input est connectée,un pointeur sur le thread utilisé par cette output et un pointeur sur le client correspondant.

- Le type output réseau contient un pointeur sur output, un descripteur de socket pour savoir avec qui cette output est connectée,un pointeur sur le thread utilisé par cette output et un pointeur sur le client correspondant.

4.3.2 Fonctions "communes"

Le client et le server ont besoin de quelques fonctions en plus que celle qui ont déjàă été crées pour la premiere soutenance :

- Fonction waitsock_data : cette fonction bloquante attends que que le buffer soit rempli avec un certain nombre d'octets grâce àă la fonction systême ioctl. Cette fonction est utilisée dans la majoritée des fonctions que nous avons présenté précédemment.

- Fonction string_send : étant donné qu'on ne peux pas envoyer directement des chaînes par le reseau, on a dû créer cette fonction. Cette fonction prend un char* et un descripteur de socket en paramètres, envoie un premier entier qui représente la longueur de la chaîne, puis la chaîne elle même.

- Fonction string_reception : étant donné qu'on ne peux pas recevoir directement des chaines par le reseau, il a fallu implémenter une fonction réalisant cela. Cette fonction prend un descripteur de socket en paramètre, attend un premier entier qui représente la longueur de la chaiîne, puis apelle la fonction waitsock_data en lui passant la longueur de la chaine en paramètre. Une fois la chaine chargée dans le buffer, on apelle la fonction recv pour recevoir notre chaîne que l'on retourne ensuite.

4.3.3 Traitement des messages réseau

Le client est le "maître" du cluster et, a donc besoin de pouvoir ordonner aux server d'effectuer les actions que le client aura commandé dans patchwork13. Pour se faire, le server possède un switch pour traiter les messages qu'il reçoit.

On a du créér une fonction waitsock_data qui est une fonction bloquante qui attend que le buffer soit rempli d'un certain nombre d'octets. Dans le `switch` cete fonction est utilisée pour recupérer un premier entier, correspondant a l'identifiant du message qui va suivre. En fonction du `case` dans lequel l'entier entre, il demande à une des fonction de s'éxécuter. Les fonctions qui ont été realisée sont :

- create_patch_client_of_server, qui demande à un server de creer un patch en lui envoyant la classe du patch ainsi que son nom dans la classe. Une fois fait, le server renvoie au client l'adresse du patch qu'il a créé, ceci afin que la liste des patchs par server, maintenue par le client, soit á jour.

- destroy_patch_client_of_server, qui detruit un certain patch et efface toutes les input/output connectés correspondants à ce patch dans les listes d'outputs/inputs du client du server correspondant.

- int connect_patch_local_client_of_server, qui connecte localement une output et une input appartenant à des patchs locaux.

- connect_patch_distant_client_of_server, qui demande à un server de connecter une des output d'un de ses patch à une input d'un patch lancé sur un autre server. Pour se faire, une connection TCP est etablie entre les deux server avec un thread de chaque côté pour gérer les communications entre ces inputs/outputs. À travers cette connection, le premier client envoie au second une demande de bind_patch_input_client_of_server ce qui a pour effet de tenir a jour les listes d'input pour le second server. Ensuite, le premier client met à jour sa liste d'outputs.

- ask_patch_start_client_of_server, sert à demarrer un patch à un certain temps donné.

- ask_patch_stop_client_of_server, sert à stopper l'éxécution d'un patch.

4.3.4 Messages du Client

Au niveau du Client(le pc principal sur le reseau celui oû est l'utilisateur) il faut envoyer les messages permettant de réaliser les actions. Pour discuter avec les Serveurs, le Client utilise des messages prédéfinis que l'on envoit au different protagoniste en jeux. Pour cela on utilise les messages préfédinis dit précédement dans la réception de message donc : «créé patch», «détruire patch», ...A partir de ça, chaque fonction a son message et on prend en paramêtre ce que le message à besoin. Puis on envoit le tout, il y a 2 fonctions aidant à cela, une envoyant un entier l'autre des chaines de charactères, et puis des “send” normal pour les structures classiques.

4.4 Ce qu'il reste à faire

Nous avons quasiement fini le cluster et sommes donc en avances sur les prévisions. Cependant il reste une fonction majeure a coder : `ask_pump_patch`, qui aura pour rôle de faire la remontée recursive aux pères de la demande de pump. Pump sur un patch a pour effet de le faire calculer ce dernier en pompant ses inputs.

A cette soutenance, niveau cluster, il est possible de lancer différents patchs sur différents servers et d'actionner ses patchs. Par exemple, il est possible de lancer un patch son sur un ordinateur, un patch opengl sur un autre et de faire fonctionner les deux patchs en même temps.

Cette fonction nous permettra donc de clore le cluster et de permettre un fonctionnement total entre les patchs repartis sur differents server.

Chapitre 5

Librairie de patches SDL

5.1 OpenGL

5.1.1 Présentation générale

La partie OpenGl du projet permettra aux utilisateurs d'avoir des patchs de base leur permettant de faire de la 3d sans se prendre la tête. Bien évidement l'openGl est roi dans ce domaine et surtout en accord avec notre projet qui doit rester portable. Pour les mêmes raisons nous avons choisi d'utiliser la SDL pour accompagner l'implémentation de l'OpenGl. C'est une librairie multiplateforme permettant de manipuler pas mal de choses qui pourrait être utile dans Patchwork13 plus tard (accés au clavier, souris, son, joystick).

5.1.2 Fonctionnement et Démarrage

Il a fallu découper les possibilitées d'OpenGl sous forme de patchs disponible par la suite. Toutes les fonctionalitées n'ont pas été transcrites pour simplifier le travail et aussi sans en faire pour que ce soit inutile, OpenGl va très loin dún point de vue technologique (Vertex Shader, Volumetric Frog, ...). Seuls des patchs de base ont donc été implémenté ce qui reste suffisant pour faire de la 3D correcte tout de même. Pour implémenter en patch les diverses capacités j'ai dÃż comprendre comment marcher le noyau du projet puis l'écriture de patch pour cela j'ai écrit dans la librairie standard un petit patch permettant de faire le calcul d'une puissance positive et négative, j'ai beaucoup appris sur ce simple exemple. Ensuite je me suis donc logiquement intéressé au manniement de l'OpenGl pour cela j'ai lu les tutorials de Nehe (http ://nehe.gamedev.net) Ãă un stade un peu plus avancés de ce que je pensais faire pour mieux analyser la librairie et ne pas bloquer l'intégration d'éléments futurs dans le logiciel, faute de mauvaise structure. Après avoir réalisé ça j'ai pu rapidement tout transcire.

5.1.3 Patch Opengl

Surface

J'ai commencé par faire l'initialisation de la surface avec toutes les lignes de codes que cela implique. Au passage j'y ai intégré les paramêtres me permettant pour plus tard d'utiliser les fonctionalités que je voulais. L'initialisation de la

surface OpenGl se fait à partir d'un êvénement "start". Puis l'enchainement des fonctions et des paramêtres se déclenchent : SDL_init, paramêtrage de la video, initialisation de la surface, redimensionage de la fenetre et déclenchement de la boucle d'affichage. Dans la boucle il y a une fonction permettant de savoir si l'utilisateur à appuyé sur les touches "Escape" ou "F" pour soit quitter le programme soit mettre la fenêtre OpenGl en "Plein Ecran" ou pas (respectivement). Le "pump" du patch permet juste de faire un swap des buffers de rendu openGL.

Forme et déplacement

Ensuite j'ai codé les patchs triangle et quad prenant en paramêtre un entier indiquant leur taille simplement ensuite je fait un "glBegin(GL_TRIANGLES)" ou "glBegin(GL_QUAD)" et j'obtiens une forme plaçant les points en fonction de la taille. Pour pouvoir placer ces triangles où l'on veut j'ai codé un patch permettant de translater, soit il prend en paramêtre, x, y et z, lui permettant de se déplacer lá où l'on veut puis hop on dessine sa forme on se redéplace et paf encore une autre forme. Parfait mais il faut ensuite inclure la notion de 3d dans tout ça. Le patch rotation est là pour ça, il prend en paramêtre l'angle plus l'axe de rotation avec x, y, z. Par exemple si on veut faire une rotation du haut vers le bas on va láppeler avec les z positif et le reste a zéros sauf l'angle bien sûr. A partir de ça on peut créer des formes en 3d en calant bien comme il faut les facettes des objets (cubes, pyramides,...). Un autre petit patch dans le même style est là aussi pour remettre "la vue" au point zéros.

Attrayance : Couleur, Transparence, Lumiéres et Textures

Voilà c'est déjà bien mais on peut rendre cela plus attrayant avec de la couleur et de la transparence par exemple. En faite ça se passe un peu comme si on modifait une variable globale qui serait la couleur courante. Le patch prend en paramêtre les entrées de la fonction qui sera appellé pour modifier cette variable locale soit à la norme du rgb, donc le rouge, vert et bleue en floatant, j'ai ajouté un paramêtre alpha pour la transparence donc sans avoir la possibilité de faire sans. Le patch sera donc lancé avec alpha à 1 si on veut une application de la couleur opaque.

Comme autre "gadget", j'ai implémenté une lumiere globale. On lui passe juste sa position (x,y,z). Je dis globale parce que l'on peut quén mettre une, de plus les normales des faces n'ont pas été implémentées donc la lumiere se réparti un peu n'importe comment sur les faces...

Pour texturer les diverses faces, il y a plusieurs façons. Pour rendre cela plus souple pour les utilisateurs du logiciel, nous avons décidé de traiter les textures comme des surfaces que l'on collerait donc sur les facettes des formes que l'ont crée. A partir d'un "start" du patch on charge la texture propre au patch (soit il faut lancer un patch par texture) dans une variable à partir d'une image qu'on donne en paramêtre au patch. Puis il faut définir des vecteurs pour cadrer la pose. Pour que se fasse j'ai cré un type vect4f(x,y,z, profondeur) permettant de simplifier le procédé de paramêtrage. Comme on applique un plan il y a 2 vecteurs un pour les abcisses l'autre les ordonnées. Donc pour l'utiliser on applique une rotation tout pareil que la facette que l'on va créé et on appelle ce

patch qui en pompant va sélectionner la texture en mémoire et l'appliquer telle un rétroprojecteur. Voilà donc un avec ces petits patchs on peut déjà faire une structure complexe texturée qui bouge et qui soit transparente par exemple, soit une réalisation sympathique pour une démo avec de la musique techno...

5.2 Patch SDL_Audio

5.2.1 Présentation générale

L'utilisation d'entrées/sorties sonores est necessaire dans la conception de patchs. Pour ce faire, nous avons décidé d'utiliser la librairie audio de SDL.

En effet, elle offre de grandes possibilitées dans le traitement de son et donc dans la créations de patchs. On peut dés à présent, initialiser un son et le jouer. On peut completement contrôler le flux de données lu dans un sample. On a même la possibilitée de creer les samples, mixer deux samples, controler la stereo...Ce qui nous permetra par la suite de créer des patchworkds de création sonore grâce à de multiples patchs Audio.

SDL premetant aussi l'utilisation d'opengl, son utilisation nous permet dêtre complètement compatible.

5.2.2 Fonctionnement et Démarrage

L'apprentissage de l'utilisation de SDL_Audio passe beaucoup par la documentation. En effet, les tutoriels et autres aides sont assez sommaires et ne nous apprend pas a gérer les buffers et à manipuler les types de données specifique a SDL. Afin d'arriver à notre resultat, nous avons été à tatons et a dissection de documetations.

5.2.3 sound_out

Comme tous les patches, on peut voir apparaître ce patch dans l'interface graphique de GTK.
L'initialisation et le lancement d'un son est compose de plusieurs étapes :

L'initialisation

Elle contient les initialisations classiques des patchs et a en plus une initialisation de SDL_Audio. Elle se fait par l'ajout d'un paramêtre SDL et d'une structure capable de gerer a la main les buffers. Ces deux sont initialisé et prêts à accueillir des données afin de les traiter et de les jouer.

Pump et traitement des donn'ees

Tout commencera lors du premier pump. Entre ce premier pump et tant que le premier buffer n'est pas rempli, on ajoute les entree à ce buffer. Une fois qu'il est chargé, on lance la lecture de ce buffer et on alterne entre les deux buffers. Les donnees arrivant en entrée sont alors traitées et envoyées dans le buffer non lu. La sortie du patch est reliée au temps du ptchwork et fait avancer le temps.

Stop

On arrête la lecture, SDL_Audio et on libère le paramertre SDL et les buffers ;

5.2.4 Ce qui est prevu

Grâce à la création de ce patch de sortie audio, il ne nous reste plus qu'a créer des patches capables de ajouter et de modifier des effets sur la sortie sonore ou tout simplement sur les samples d'entrée.

Chapitre 6

Interface graphique GTK+2

L'interface graphique a beaucoup progressé depuis la dernière soutenance. Il est désormais possible de créer des patches, de les déplacer et même de les connecter entre eux, le tout à la souris.

6.1 Glisser - déposer

Toutes ces opérations peuvent en fait se rapporter à des glisser - déposer (*drag and drop* en anglais). Nous avons donc fait grand usage de la méthode fournie par GTK pour effectuer des glisser - déposer. Malheureusement cette méthode est très très (très) mal documentée (contrairement au reste de GTK) et il a fallu avancer à tâtons, d'autant plus que la méthode est assez complexe et implique d'utiliser plusieurs signaux pour communiquer entre le widget source et le widget d'arrivée.

Pourtant, une fois le fonctionnement apréhendé, la methode est très efficace. C'est pour cela qu'elle à été choisie pour effectuer les trois opérations suivantes :

6.1.1 Création d'un patch

On peut créer un patch en le prenant à la souris dans l'arbre de la fenêtre principale, et en le déposant dans le fenêtre d'un patchwork. La liaison avec la fonction du noyau pour créer un patch est faite, la création d'un patch dans l'interface graphique et dans le noyau est faite simultanément.

La création du widget d'un patch est faite via un fichier glade qui contient ce qui est commun à tous les paches. Il faut ensuite appliquer le nom du patch et créer les widgets correspondant aux entrées et aux sorties du patch créé.

6.1.2 Déplacement d'un patch

Le déplacement d'un patch dans le patchwork se fait lui aussi via un glisser - déposer depuis le nom du patch, en restant dans la fenêtre du patchwork. On récupere ainsi la nouvelle position du patch, que l'on déplace. On enregistre la position du patch car on en a besoin pour dessiner les flèches qui relient les patches entre eux.

6.1.3 Connection des patches

La connection d'une sortie d'un patch se fait en la tirant à la souris jusqu'à une entrée d'un autre patch. La connection vérifie si les type de données sont compatibles, si les patches ne sont pas déjà connectés et si tout est valide effectue la connection. On enregistre alors la connection dans une liste chaînée, pour pouvoir afficher, detruire ou effectuer d'autres actions sur les connections.

6.2 Dessin des connections

6.3 Drag & Drop

6.4 Affichage

6.4.1 Le Patchwork

L'affichage du patchwork a changé...
En effet, un contexte graphique a été inséré. Il est créé grace à la lib cairo et nous permet de dessiner, d'afficher des fenêtres, et de pouvoir connaître les caracteristiques de chacun par rapport aux autres. On peut toujour voir apparaitre la barre de contrôle permetant de demarrer, mettre en pause, arrêter un patchwork. Ces fonction sont limitées à tester si les procedures des patches sont correctement executées.

6.4.2 Cairo et GTK

La dernière version de Cairo, est insérée dans GTK. C'est à dire que la lib est directement gérée par GTK. Mais afin de pouvoir profiter de ces avancées technologiques, il nous a fallu aquérir la dernière version de GTK. Les classiques 'emerge' ou 'apt-get' de linux nous le permet, mais sous mac os x, c'est différent. En effet, l'installation de 'fink' (similaire a 'apt-get') n'a pas suffit, il nous a fallut installer 'darwinport' pour pouvoir bénéficier d'une version assez recente. L'affichage est à present présent sur mac os x et linux. le problème étant resolu, on peut desormais jouir d'un bel affichage muti-os.

6.4.3 Les Patchs

Les patches sont enfin aparus ! Hé oui, on peut voir maintenant apparaître les patchs et leurs contenus dans la fenetre du patchwork. On peut même la faire glisser de la liste des patches dans le patchwork grâce au drag & drop.

La fenêtre du patch est composée de plusieurs parties :
- Le titre
  Boite contenant le nom du patch.
- Les entrées / sorties
  Crées dynamiquement à la création du patch dans le patchwork. Les checkbox représentant les entrées et les sorties sont liés à leur nom et une sortie est reliable à une entrée à la souris grâce au drag & drop.

6.4.4 Les Plêches

Les flèches representent les liaisons entres les entrées et les sorties des patches d'un patchwork. Elles sont dessinées lors d'un déplacement ou d'un drag & drop.

Afin de gérer l'affichage de toutes les flèches, nous avons créé une liste de connections entre deux patches. Elle est mise à jour à chaque nouvelle connection et est parcourue quand on rafraîchi la fenêtre du patchwork.

Le placement des points de départ et d'arrivée n'a pas été une mince affaire. On a eu un petit probleme quant à la position de la checkbox dans sa boite, les insertions des checkbox étant dynamiques, on a du passer par des fonctions d'appel au parent et un certain décalage est venu s'ajouter pour pouvoir enfin avoir des fleches bien cadrés.

6.5 Pour la prochaine soutenance

Il reste juste à implémenter dans l'interface la suppression dúne connection et la suppression d'un patch. Ce sont deux fonctions assez triviales et ne devraient pas poser problème.

Si l'importation d'un fichier XML est terminée pour la prochaine soutenance, il faudra également ajouter la fonction dans l'interface du patchwork...

Pour prendre de l'avance il est possible que nous commencions à inclure certaines fonctions du cluster dans l'interface graphique comme nous l'avions fait à la première soutenance avec la liste des serveurs.

Chapitre 7

Conclusion

Le projet Patchwork13 se dessine de plus en plus vers le résultat recherché et nous sommes en avance sur le planning ce qui nous rend confiants sur la finalisation future du projet.

Au niveau de l'interface, il est désormais possible de créer des patchs dans le patchwork à la souris et de les relier entre eux à la souris, ceci couplé avec un effet visuel de fléches représentant les liaisons créés.

Du côté cluster, on peut faire créer des patchs à un server, les démarrer, les arrêter et les détruire. En resumé un client peut par exemple demander à un server de dèmarrer un patch son et donc par exemple de pouvoir faire un concert en salle machine où chaque machine jouera un instrument différent, tout cela commandé par le client.

Tout comme a la première soutenance, le groupe a fait preuve d'une bonne cohesion ce qui nous a permi de coder plutôt efficacement (oui plutôt car des fois on avait du mal a cause a cause de ... enfin tout le monde s'en doute ... *sifflotte*).

Il nous reste donc à finaliser le cluster pour que l'interopérabilité entre les patchs soit complete et de lier le cluster Ãă l'interface afin d'obtenir le projet complétement utilisable. Suivront de nombreux patchs pour bien illustrer les possibilités du projet.

Sur ce, rendez-vous à la prochaine soutenance !