Mapping modulaire de processus polyphoniques

Vincent Goudard, Hugues Genevois

MAPPING MODULAIRE DE PROCESSUS POLYPHONIQUES

Vincent Goudard
Collegium Musicæ
Sorbonne Universités
goudard@lam.jussieu.fr

Hugues Genevois
LAM - Institut Jean Le Rond d'Alembert
genevois@lam.jussieu.fr

RÉSUMÉ

Nous présentons le système MP, un protocole et un ensemble d'outils facilitant la connexion modulaire de processus polyphoniques. Le but de cette librairie est d'améliorer la modularité en conservant les blocs individuels de traitement polyphonique indépendants du mapping général qui a lieu dans le design de l'interaction d'un instrument de musique numérique. Le protocole MP est fondé sur un paradigme à trois états permettant la modulation expressive de tout paramètre. Une stratégie originale est proposée pour le groupement hiérarchique d'événements en spécifiant quels *événements invités* seront autorisés à modifier les paramètres d'une voix de polyphonie affectée à cet événement. Une revue préalable des principaux protocoles qui ont inspiré ces développements permettra d'en expliciter les enjeux. Quelques exemples seront donnés pour expliciter les possibilités offertes par cette stratégie. Les développements présentés ici sont implémentés dans le logiciel Max mais la logique sous-jacente reste applicable dans d'autres systèmes utilisant une logique de communication asynchrone.

1. MOTIVATIONS

1.1. Contexte

Les instruments de musique numérique nous mettent face à une équation bancale. D'un côté quelques flux de données issus de capteurs rendent péniblement compte de la finesse du geste physique, de l'autre la possibilité (et le désir) de produire une musique plus complexe qu'aucun instrument acoustique ne peut le faire.

La réponse à cette équation réside dans la question centrale de ce que l'on a coutume de nommer le *mapping*, c'est à dire la relation de correspondance entre valeurs issues de capteurs et paramètres de synthèse. Cependant, et malgré sa popularité[1], le terme de *mapping* semble assez peu représenter la complexité de ce qui n'est pas une simple mise en relation et nous préférerions parler d'un *design d'interaction*, c'est à dire d'une programmation des relations entre gestes et sons faisant intervenir modèles dynamiques, scénarios évolutifs, fragments de partitions et réglages d'un nombre extrêmement élevé de variables.

Dans un article précédent [3], les auteurs ont proposé l'utilisation de *modèles intermédiaires dynamiques* pour enrichir le geste capté et améliorer l'ergonomie des instruments de musique numériques. Cependant, un des facteurs critiques rencontré lors de ces développements se situait dans la manière de faire communiquer différents modules polyphoniques[2] entre eux. Le défi était de rendre cette communication polyphonique, tout en gardant la trace de l'identité et l'ordonnancement des événements permettant que le calcul s'effectue correctement.

1.2. Revue des protocoles de contrôle

1.2.1. Remarques générales

Les protocoles de contrôles que nous envisageons ici sont asynchrones, fondés sur l'idée que les systèmes musicaux envisagés dans les lutheries actuelles sont des systèmes complexes composés d'éléments hétérogènes et intégrant notamment des interfaces hardware elle-mêmes asynchrones. Bien que la synthèse audio soit un processus synchrone, le design global d'un DMI[3] est le plus souvent un système « globalement asynchrone, localement synchrone » tel que défini par Chapiro [1].

Un certain nombre de protocoles dédiés au contrôle temps-réel de la synthèse numérique ont vu le jour depuis les années 1980. Au delà de proposer des solutions techniques au problème du contrôle, ces protocoles sont porteurs d'un modèle implicite représentant les objets en présence dans l'interaction geste/son. Une brève revue montrera comment ceux-ci se sont progressivement ouverts, à mesure que les capacités de calcul se sont accrues et que la notion même d'instrument s'élargissait à de nouveaux champs tels que les installations sonores ou les applications musicales interactives.

[1] Entre 2001 et 2015, le terme "mapping" a été employé dans plus de 750 articles de la conférence NIME (New Interfaces for Musical Expression) alors que "design d'interaction" (interaction design) n'était employé que dans 160 articles.

[2] Par « module polyphonique », on entend ici des processeurs traitant simultanément plusieurs flux de données de contrôle de même nature en parallèle, e.g. le filtrage des points de contact sur une interface multi-touch ou encore la modulation des différentes notes d'un accord.

[3] Digital Musical Instrument.

1.2.2. MIDI

Le MIDI a fêté son trentième anniversaire en restant le protocole le plus répandu pour le contrôle de la synthèse audio. La profusion de nouvelles interfaces et applications l'auront tout juste fait évoluer pour permettre la prise en charge de nouvelles technologies de réseau (rtpMIDI[4]) ou de nouvelles interfaces (MPE[5]).

Les limitations du MIDI ont pourtant été identifiées peu de temps après son apparition [5][7][9], notamment :

- sa précision et son espace de nommage sont limités,
- l'identifiant d'une note est assimilé à son (éventuelle) hauteur,
- l'état actif d'une note est assimilé à sa vélocité,
- la modulation individuelle des notes est fastidieuse,
- sa nomenclature fait référence aux instruments acoustiques.

Le MIDI élude une partie de la question du mapping en reliant intrinsèquement le geste à la production sonore à travers le concept de *note MIDI*[6] qui assimile les deux côtés de l'interaction : la notion de *vélocité* se rapportant au geste et celle de *pitch* au son.

1.2.3. ZIPI

En 1994, Zeta Instruments et le CNMAT[7] proposèrent ZIPI [5] pour dépasser les limitations du MIDI. ZIPI fait ainsi la distinction entre note, hauteur, canal et vélocité, augmente la précision des données, introduit des messages de modulation par note, la possibilité d'un réseau en étoile (plutôt que le chaînage linéaire MIDI) ainsi qu'une méthode d'interrogation des instruments connectés.

ZIPI propose également une organisation hiérarchique à trois niveaux héritée d'une classification traditionnelle où des *orchestres* sont fait de *familles d'instruments*, composées d'*instruments*, eux-même fait de *notes*. ZIPI introduit enfin deux espaces de nommage distincts pour la description du geste d'une part et de la synthèse audio d'autre part.

Malheureusement, le public cible des fabricants et utilisateurs de synthétiseurs hardware n'était pas prêt pour un tel changement alors que l'avènement du protocole IEEE 1384 cette même année palliait le faible débit de données du MIDI[8].

1.2.4. Open Sound Control

En 1997 au CNMAT, un groupe incluant d'anciens concepteurs de ZIPI ré-utilisa la recherche menée pour publier le protocole *Open Sound Control* (OSC) [10], motivé par le besoin de répondre à l'évolution des technologies de réseau et l'extension des types de données alimenté par l'utilisation grandissante de logiciels comme Max. En proposant une syntaxe intelligible et facile à utiliser, OSC a connu un certain succès : il a été adopté par un certain nombre de logiciels et interfaces hardware[9] et utilisé pour définir d'autres protocoles tels que GDIF[10], TUIO[11] ou encore la librairie « o. »[12].

Cependant, sa relative lourdeur en terme de débit comparé au MIDI [2] et une absence de nomenclature rendant fastidieuse les branchements plug'n play l'ont pour l'instant privé d'une adoption par l'industrie et le grand public.

1.2.5. TUIO

Dans cette brève revue, il faut mentionner TUIO [4] (fondé sur OSC) comme le premier protocole à introduire un indice incrémentiel pour identifier de manière unique des événements dynamiques et éphémères tels que les touchés de doigt sur une interface utilisateur tangible (TUI).

Il se distingue également de la logique des événements MIDI[13] en proposant une solution simple et pratique pour résoudre l'incertitude d'arrivée des messages envoyés sur UDP[14] et consistant à envoyer systématiquement la liste des événements actifs.

1.2.6. MPE

La dernière évolution du MIDI a été motivée par la commercialisation récente d'interfaces dites *expressives*[15], permettant la modulation de chaque note jouée. Bien qu'elles ne soient pas les premières interfaces permettant un tel contrôle, un effort conjoint a été entrepris par plusieurs fabricants pour définir standard nommé *Multidimensional Polyphonic Expression* (MPE).

Cependant, cette évolution n'est qu'une normalisation de l'usage des canaux MIDI actuels permettant un tel contrôle dans le cadre existant.

---

[4] Encapsulation du MIDI dans des messages RTP permettant une communication sur des réseaux ethernet et WiFi.

[5] Multidimensional Polyphonic Expression : convention pour l'assignation automatique de chaque nouvelle note à un nouveau canal MIDI. (Spécifications en cours d'élaboration.)

[6] Une note MIDI est composée d'une valeur de *pitch* et d'une valeur de *vélocité* associées à *canal* MIDI.

[7] Center for New Music and Audio Technologies, University of Berkeley, California.

[8] Le débit d'un bus MIDI était jusqu'alors de 31,25 kbit/s en connection DIN uni-directionnelle ; l'IEEE 1384a (alias "firewire") proposait jusqu'à 400Mbit/s tout en étant bi-directionnel.

[9] Comme le Lemur, le Monome ou l'Ethersense.

[10] GDIF : Gesture Description Interchange Format.

[11] TUIO : Tangible User Interface Protocol.

[12] « Oh dot » : librairie Max développée au CNMAT.

[13] … dont l'utilisation de messages distincts pour les note-on et -off peut produire des notes qui restent "bloquées" si un message note-off est perdu.

[14] User Datagram Protocol.

[15] Citons le LinnStrument, Roli Seabord, Haken Audio's Continuum, Eigenharp Alpha, Madrona Labs' Soundplane, le KMI K-Board Pro4 ou prochainement Joué.

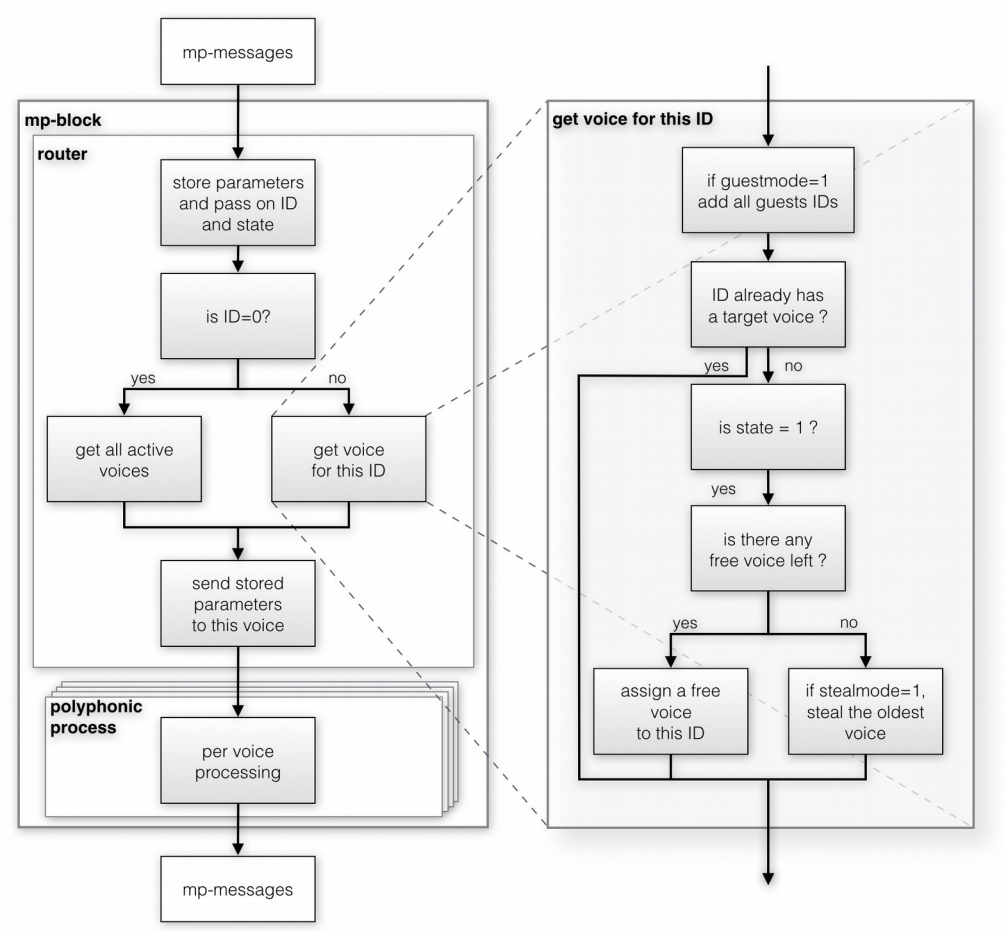

Figure 1. *Schéma de fonctionnement d'un mp-block.*

2. LE SYSTÈME MP

MP ("Modular Polyphony")[16] est un *framework* composé de modules de traitement polyphoniques et d'un protocole de messages permettant leur adressage. Il s'inspire de certaines des idées présentes dans les protocoles pré-cités. En particulier, il reprend un concept général du MIDI qui conçoit le contrôle musical temps-réel comme une séquence temporelle de *gestes* ayant un début et une fin. Il emprunte aussi à ZIPI l'idée d'un découplage entre identifiant, hauteur, vélocité, canal ainsi que la nuance entre l'activation d'une note et sa modulation. Par ailleurs, comme ce framework est destiné à une lutherie expérimentale et exploratoire, MP laisse le typage et l'espace de nommage des paramètres ouverts, sans le restreindre à une nomenclature arbitraire. Néanmoins, il propose une syntaxe plus restrictive qu'OSC permettant une gestion cohérente et facilitée de la polyphonie. Enfin, MP permet l'association dynamique d'événements entre eux, de tel sorte qu'il soit possible de contrôler les paramètres par groupes (et sous-groupes).

MP se compose de blocs de traitement polyphonique nommés *mp-blocks*, communiquant par des messages asynchrones nommés *mp-messages* et représentant des objets temporels abstraits nommés *mp-events*. Les sections suivantes décrivent ces éléments en détail.

2.1. mp-events

Un *mp-event* est un objet temporel abstrait qui peut être traité par des *mp-blocks*. Il est défini par un ensemble de *mp-messages* (figure 1). Ces messages sont composés de paramètres de contrôle précédés par un identifiant unique propre au *mp-event*. Le format de message est minimaliste et tous les messages utilisent la même syntaxe : un identifiant unique, un nom de paramètre suivi d'une liste de valeurs. Par exemple :

- [42 pitch 112] : le *mp-event* #42 règle le paramètre de pitch à la valeur 112,
- [123 scale 0 2 4 5 7 9 11] : *mp-event #123* définit une gamme diatonique.

Deux noms de paramètre sont réservés pour un usage particulier : *state* et *guests*. Ils seront détaillés dans les prochaines sections.

[16] Disponible sur :https://github.com/LAM-IJLRA/ModularPolyphony

Nous verrons également qu'un *mp-event* peut suivre plusieurs chemins de traitement en parallèle et être fusionné avec d'autres *mp-events*.

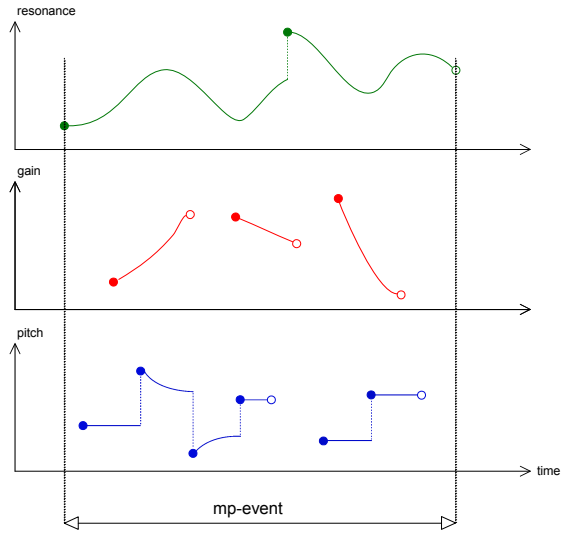

Figure 2. Un *mp-event* est constitué d'un ensemble de paramètres associés à un identifiant, et passés à un *mp-block* avec un état 1 (début de modulation), 2 (mise à jour de modulation) ou 0 (fin de modulation). Les cercles pleins représentent les états 1, les cercles vides les états 0 et les lignes continues les états 2.

2.1.1. Identifiant

L'identifiant unique ("ID") sert à identifier un *mp-event* tout au long de ses traitements. Le *mp-event* peut provenir d'un capteur physique (e.g. une touche de clavier) ou d'une source virtuelle (e.g. représentant un contact sur une TUI ou un produit par un algorithme génératif).

L'ID est présent dans tous les *mp-messages* pour éviter toute erreur de routage dans le cas où un *mp-block* serait attaqué par plusieurs sources de *mp-events* indépendantes.

2.1.2. Etat

Le message d'état est un message réservé qui remplit deux missions :

- il sert d'horloge asynchrone en déclenchant l'envoi de paramètres au processus,
- il spécifie la manière dont le processus doit interpréter ces paramètres.

Les paramètres peuvent ensuite être interprétés de trois façons :

- state 1 : début de modulation de paramètre,
- state 2 : mise à jour de paramètre,
- state 0 : fin de modulation de paramètre.

Le modèle musical sous-jacent envisage un *mp-event* comme la modulation d'un ensemble de paramètres et prend en compte les phénomènes transitoires qui peuvent apparaître au début et/ou à la fin d'une modulation[17]. Ces discontinuités peuvent causer des réponses non-linéaires, trop rapides pour être contrôlées manuellement et parfois mieux traitées séparément.

Au niveau du processus, ces états peuvent correspondre à l'initialisation de variables internes, l'activation d'un lissage (*portamento*) entre deux valeurs consécutives, le déclenchement d'un processus transitoire spécifique (e.g. l'attaque d'un son), etc.. Tout paramètre peut donc être envoyé à un *mp-block* en spécifiant s'il doit être considéré comme le début, la continuation ou la fin d'une phrase de modulation[18].

Ce modèle à 3 états semble correspondre par ailleurs aux intentions[19] de la MMA[20] qui a annoncé un message de *note-update* dans le futur protocole MIDI-HD.

Notons toutefois que dans notre cas, le message d'état peut être rattaché à n'importe quel paramètre, ce qui a des conséquences différentes du MIDI en ce qui concerne l'allocation de voix. Alors que le premier message *state 1* reçu pour un ID causera l'allocation d'une voix dans le *mp-block,* le message *state 0* ne libérera pas nécessairement cette voix, cette décision revenant au processus en cours, comme nous le verrons plus loin.

2.1.3. Guest-list

La spécification MP ne suit pas d'organisation hiérarchique telle que les canaux MIDI ou les familles d'instruments de ZIPI. A la place, elle laisse la possibilité à tout *mp-event* de déclarer une *liste d'invités* (*guestlist*) à la volée. Ces invités pourront avoir accès à la voix allouée à un *mp-event* et contrôler ses paramètres. Cette fonctionnalité nous offre une solution flexible pour le groupement d'événements, permettant un nombre arbitraire de niveaux hiérarchiques, sans pour autant être limité par une relation de subsumption.

Cette fonctionnalité peut être utilisée dans le cas de *mp-blocks* génératifs, où l'ID du *mp-event* peut être ajouté à la *guestlist* des *mp-events* « enfants ». Ceci permet une modulation cohérente de plusieurs voix associées à des *mp-events* générés par une même source. Des exemples concrets de cette situation sont les pistes MIDI ou la hiérarchie orchestre/famille/instrument/note proposée par ZIPI. Ils correspondent à un mapping

[17] Un exemple évident est l'attaque d'un son, mais en ce qui concerne un processus non-sonore comme le filtrage de données, cela peut concerner l'initialisation du filtre.

[18] La confusion entre le *pitch* et l'identifiant de note dans le protocole MIDI rend le résultat de la même opération incertain : alors que certains synthétiseurs re-déclencheront la même voix, d'autres alloueront une nouvelle voix et attendront le même nombre de *note-off* qu'il y a eu de *note-on*.

[19] Rapporté par (dernier accès janvier 2017) : http://www.synthtopia.com/content/2013/01/20/midi-manufacturers-testing-new-high-definition-midi-protocol/

[20] MIDI Manufacturer Association

divergent dans le schéma proposé par Rovan, Wanderley, Dubnov et al. [8].

Un *mapping convergent* est également possible : plusieurs *mp-events* peuvent être déclarés comme *guests* d'un *mp-event* tiers. Par exemple, un objet graphique virtuel (représenté par un *mp-event*) peut être « touché » par plusieurs *mp-events* représentant des contacts sur une surface TUI.

2.1.4. Master-ID

Le paramètre *guests* ne nous laisse toutefois pas un accès aisé à l'ensemble des voix de polyphonies d'un *mp-block*. A cette fin, un identifiant spécifique : "0" permet ce contrôle global. Il revient implicitement à considérer que le *mp-event #0* (nommé *master-ID*) fait systématiquement partie de la *guestlist* de tout *mp-event*. Dans le cas où les *mp-events*, ses *guests* et le master-ID tentent de modifier les mêmes paramètres, le *mp-event* hôte conservera la priorité sur les *guests* et le *master-ID*.

2.1.5. Ordonnancement des mp-messages

Le cycle de vie de la voix d'un *mp-event* suit la séquence suivante de *mp-messages* :

1. envoi des paramètres de début de modulation,
2. envoi du message state 1,
3. envoi des paramètres de modulation,
4. envoi du message state 2,
5. envoi des paramètres de fin de modulation,
6. envoi du message state 0.

Cependant, comme la libération d'une voix ne suit pas nécessairement un message *state 0* (dans le cas où le processus a sa propre stratégie d'extinction), il est possible d'envoyer différents messages d'état plusieurs fois durant la durée de vie d'une voix, jusqu'à ce que la voix soit effectivement libérée.

2.2. mp-blocks

Un *mp-block* se compose de deux parties : le routeur et le traitement polyphonique que nous décrivons ici.

2.2.1. Le routeur

Le routeur a pour missions :

- l'allocation de voix pour un nouvel *mp-event*,
- le routage des paramètres à cette voix,
- le routage éventuel des paramètres des *guests*,
- d'enregistrer la libération de la voix.

L'ordonnancement des messages envoyés à la voix de traitement assure une synchronisation du calcul afin qu'il ne soit fait qu'une seule fois. Ainsi, la séquence suivante est envoyée par le routeur à la voix cible :

1. start X state Y : ce message permet à la voix de se préparer à traiter les paramètres à venir selon l'état Y,
2. tous les paramètres du *master-ID*,
3. tous les paramètres des *guests*,
4. tous les paramètres du *mp-event* déclencheur,
5. end X state Y : ce dernier message clôt la séquence et sert de signal d'horloge déclenchant le calcul.

L'allocation de voix est réalisée à l'arrivée du premier message *state 1* pour un *mp-event* donné. La libération de la voix intervient quand le processus de traitement indique au routeur qu'il a terminé sa tâche. Trois scénarios sont alors possibles :

- Le processus se termine dès qu'un message state 0 est reçu. Ce sera le cas, par exemple, pour un processus tel qu'une addition ou un filtre médian. Ce scénario est pris en compte de manière automatique en spécifiant un attribut "*@automute 1*" au routeur.
- Le processus déclenche son extinction à la réception d'un message *state 0*, typiquement le *release* d'une enveloppe ADSR.
- Le processus a sa propre durée et peut avoir terminé sa tâche avant ou après avoir reçu un message *state 0*. C'est le cas, par exemple, lors de la génération d'un signal audio de durée fixe comme un échantillon de percussion.

2.2.2. Le traitement par voix

Le traitement par voix est un patch chargé dans les multiples voix d'un objet Max poly~. En dehors des fonctions permettant le traitement à proprement parler, un objet nommé *mp-muter* permet de dispatcher les messages envoyés par le routeur en fonction du message d'état (tel que décrit à la section 2.2.1) et de renvoyer au routeur l'information de fin de tâche que le processus doit fournir dans le cas où il possède une extinction propre (mode *@automute 0*). Ce principe est exposé sur la figure 3, dans laquelle l'objet adsr~ vient notifier l'objet *mp.muter* de la fin de l'enveloppe.

2.2.3. Paramètres d'un *mp-block*

Les *mp-blocks* répondent à une liste de paramètres contrôlant le processus de traitement par voix. Ces paramètres sont stockés par identifiant dans le routeur jusqu'à ce qu'un message d'état soit reçu. Ce message entrainera l'allocation d'une voix (si disponible et si elle n'est pas déjà active), puis l'envoi de tous les paramètres partageant cet identifiant à cette voix.

Il est possible d'envoyer d'autres paramètres que ceux contrôlant le processus. Dans ce cas il traverseront le *mp-block* inchangés, tout en restant synchrones avec d'éventuels autres paramètres générés par le processus. Dans le cas où ce processus génère de nouveaux *mp-events*, ces paramètres peuvent automatiquement être ajoutés à chaque *mp-event* généré, en une sorte d'héritage similaire à celui opéré par la librairie « o. » décrit dans [3].

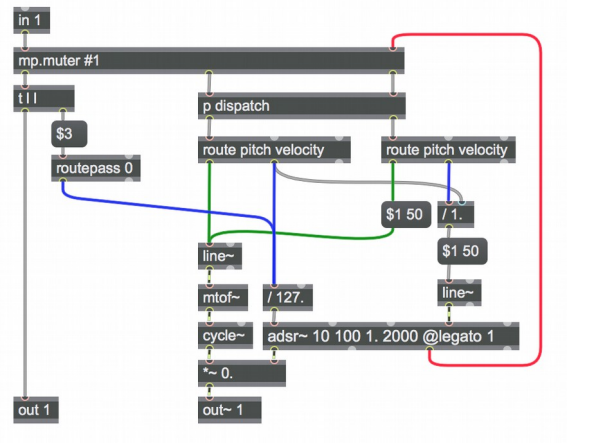

Figure 3. Exemple simple de traitement. Les paramètres de *pitch* et de *vélocité* sont différenciés selon qu'ils sont transmis accompagnés d'un état 0, 1 ou 2 afin que l'attaque d'une note s'effectue sans *portamento* tandis que les modulations de hauteur soient lissées.

3. EXEMPLES

Les exemples proposés dans cette sections donnent des cas concrets d'utilisation du système MP ; il sont implémentés dans le logiciel Max.

3.1. Exemple de mapping encapsulé

Cet exemple (figure 4) montre l'utilisation la plus simple du système MP. À partir de données TUIO issues d'une interface multitouch, on contrôle les valeurs de vélocité et de cutoff sur l'axe vertical tandis que l'axe horizontal contrôle le pitch.

Le module « *mp-embedded_mapping_demoSynth* » est constitué d'un routeur et d'un objet poly~ contenant à la fois le mapping et la synthèse. Ici, l'avantage d'utiliser MP est de bénéficier d'un système d'adressage similaire au MPE sans être limité par le typage, la précision et l'espace de nommage des données.

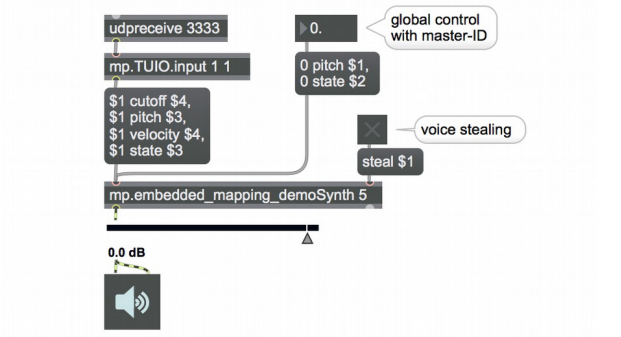

Figure 4. Mapping encapsulé : tous les processus de mapping sont inclus dans le module de synthèse.

3.2. Exemple de mapping désencapsulé

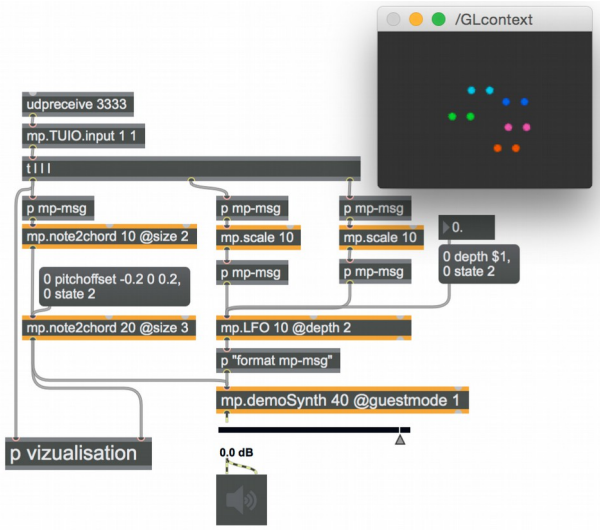

Figure 5. Mapping désencapsulé : mise à l'échelle, génération d'accords et LFO sont réalisés en dehors du module de synthèse.

Dans cet exemple (figure 5), nous avons sorti tout le mapping en dehors du module de synthèse. Examinons quelques éléments de ce patch :

- à gauche, mp.note2chord génère un ensemble de *mp-events* « enfants ». Le premier niveau en génère deux et le deuxième niveau en génère 3. Le résultat final est la génération d'un accord de six notes à partir d'un seul *mp-event,*
- à droite, le *mp-event* venant de *mp.TUIO.input* est envoyé sur deux chemins où ses valeurs sont mises à l'échelle par le module *mp-scale* pour définir la fréquence et l'offset d'un LFO[21] respectivement. Le LFO contrôlera ensuite la fréquence de coupure d'un filtre passe-bas du module de synthèse,
- nous avons deux occurrences d'un contrôle global avec le Master-ID : une pour le paramètre *depth* du LFO et une pour la transposition du deuxième étage *note2chord*,
- en dernier lieu, nous dirigeons les messages résultants de ces deux chemins vers un système de représentation graphique. La position d'objets graphiques est assignée aux valeurs de *pitch* des *mp-events* générés, tandis que leur couleur est associée au *mp-event* parent. Ceci résulte en une représentation graphique de toutes les hauteurs en tant qu'objets dont la couleur nous dit à quelle source commune (ici le contact d'un doigt sur un écran) ils sont rattachés.

[21] Low Frequency Oscillator.

3.3. Exemple de mapping many[guests]-to-one

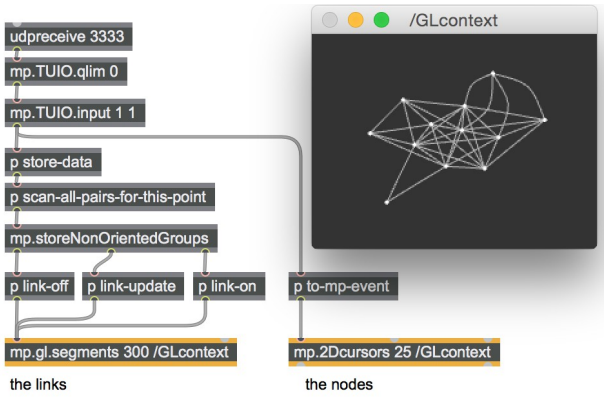

Figure 6. Les arcs sont générés quand la distance entre deux nœuds passe au dessous d'un certain seuil.

Alors que les exemples précédents nous montrent une utilisation de la *guestlist* comme hiérarchie de subsumption, ce dernier exemple fournit un exemple concret de relation *many-to-one*. Ici, les *mp-events* arrivant d'une TUI et représentant la position des doigts (les points) causeront l'apparition d'un arc si la distance entre eux passe sous un certain seuil.

Le *mp-event* « arc » ainsi généré aura les deux *mp-events* « points » dans sa *guestlist*, de telle sorte que toute modification sur les points affectera le traitement aval du *mp-event* représenté par l'arc.

4. LIMITATIONS ET OPTIMISATIONS

Séparer les différents processus de traitement d'un design d'interaction polyphonique a l'avantage de permettre une meilleure modularité, et par suite une meilleure stabilité des processus mis en œuvre qui n'auront pas à être modifiés en interne. Le choix a également été fait de ne pas s'appuyer sur une mémoire globale (e.g. pour sauver la *guestlist*) de sorte que les *mp-blocks* soient réellement indépendants et autonomes et que le design d'interaction général puisse être réparti sur plusieurs applications et/ou machines en réseau.

Cependant, cette modularité a un coût et est probablement moins efficace que d'inclure tous les traitements nécessaires dans les traitements ad-hoc. Certaines optimisations peuvent être réalisées pour des processus ne nécessitant pas de mémoire interne (e.g. une mise à l'échelle statique), pour lesquels il n'est pas nécessaire d'allouer des voix. Cela se fait cependant au prix de certaines fonctionnalités (pas d'utilisation du *master-ID* possible dans ce cas).

De plus, le framework MP a entièrement été réalisé à l'aide d'objets natifs Max et pourrait sûrement être optimisé en le portant sous la forme d'*externals* compilés.

5. CONCLUSIONS

Nous avons présenté le système MP, permettant la connexion modulaire de processus polyphoniques. Bien qu'il soit encore à un stade précoce, ce système nous permet d'explorer de nouveaux modèles de lutherie numérique, impliquant notamment des contrôleurs *expressifs* et des interfaces multi-touch ou composites.

Ces développements sont également menés pour répondre à d'autres questions telles que la représentation et l'ergonomie des processus polyphoniques que nous envisageons de discuter prochainement.

Ce travail s'inscrit dans une recherche doctorale soutenue par le Collegium Musicæ, Sorbonne-Université, que nous remercions.

6. BIBLIOGRAPHIE

[1] Chapiro, D. M. *Globally-asynchronous locally-synchronous systems*. PhD thesis, Stanford University, États-Unis, 1984.

[2] Fraietta A. « Open Sound Control: Constraints and Limitations » , *Proceedings of the New Interfaces for Musical Expressio*n, Gênes, Italie, 2008.

[3] Freed, A., Maccallum, J., et Schmeder, A. « Dynamic, Instance-based, object-oriented programming in Max/MSP using open sound control message delegation » , *Proceedings of the International Computer Music Conference,* Université d'Huddersfield, Royaume-Uni, 2011.

[3] Goudard V., Genevois H. , Doval B., et al. « Dynamic Intermediate Models for audiographic synthesis », *8th Sound and Music Computing Conference*, Padova University Press, Italie, 2011.

[4] Kaltenbrunner M., Bovermann T. , Bencina R., et al. « TUIO: A protocol for table-top tangible user interfaces » , *Proceedings of the The 6th International Workshop on Gesture in Human-Computer Interaction and Simulation*, Vannes, France, 2005.

[5] McMillen K., Wessel D.L., Wright M. « The ZIPI music parameter description language » , *Computer Music Journal*, vol. 18 (4) :52-73, 1994.

[6] MIDI manufacturers association, et al. *The complete MIDI 1.0 detailed specification*. Los Angeles, USA, 1996.

[7] Moore F. R. « The dysfunctions of MIDI », *Computer music journal*, vol. 12 (1) :19-28, 1988.

[8] Rovan J.B., Wanderley M.M., Dubnov S., et al. « Instrumental gestural mapping strategies as expressivity determinants in computer music performance » , *Proceedings of the AIMI International Workshop Kansei : The Technology of Emotion*, Université de Gênes, Italie, 1997.

[9] Selfridge-Field E. *Beyond MIDI: the handbook of musical codes*. MIT press, Cambridge, États-Unis, 1997.

[10] Wright M. , Freed A. , « Open Sound Control : A New Protocol for Communicating with Sound Synthesizers », *Proceedings of the ICMC* Thessalonique, Grèce, 1997.