Numéros / n° 7-8 - Culture du code

Â«Â Camomile, enjeux et développements d?un *plugin* audio embarquant Pure Data Â Â»

Pierre Guillot

Résumé

Cet article présente Camomile, un *plugin* audio multiplateforme compatible avec de multiples formats et permettant de charger des patchs Pure Data dans les stations de travail audionumérique. Cette présentation revient tout d?abord sur les contextes d?utilisation dans lesquels la nécessité d?un tel outil est apparue. L?objectif est de mettre en avant les principaux enjeux fonctionnels et techniques du projet. Par la suite, un état de l?art présente les différentes propositions et les évolutions dans ce domaine. Parcourir ces travaux, des premières approches des logiciels de type *patcher* embarqués jusqu?aux projets plus récents de *plugins* audio, permet notamment de clarifier les difficultés, les contraintes et les questions soulevées par ces approches afin de comprendre les choix opérés lors de la mise en ?uvre du projet Camomile. Des solutions sont alors apportées en revenant sur les principaux axes du développement de l?outil. Il s?agit de présenter les choix importants opérés au cours des différentes versions et les conséquences sur l?outil final et son usage. Enfin, le bilan et les perspectives de ce projet sont exposés.

Introduction

Camomile [(1)] est un outil permettant de créer des *plugins* audionumériques ? des modules externes destinés à être utilisés dans les stations de travail audionumériques ? à partir de patchs Pure Data [(2)] (Miller, 1997). Cet outil, embarquant le moteur de Pure Data, est lui-même un *plugin* aux formats LV2 [(3)] , VST2, VST3 [(4)] et Audio Unit [(5)] pour les systèmes d?exploitation Linux, Windows et macOS. L?utilisateur peut employer les *plugins* de Camomile comme base et leur associer les patchs de traitement ou de synthétiseur sonore élaborés dans Pure Data, afin de concevoir des *plugins* autonomes (Figure 1).

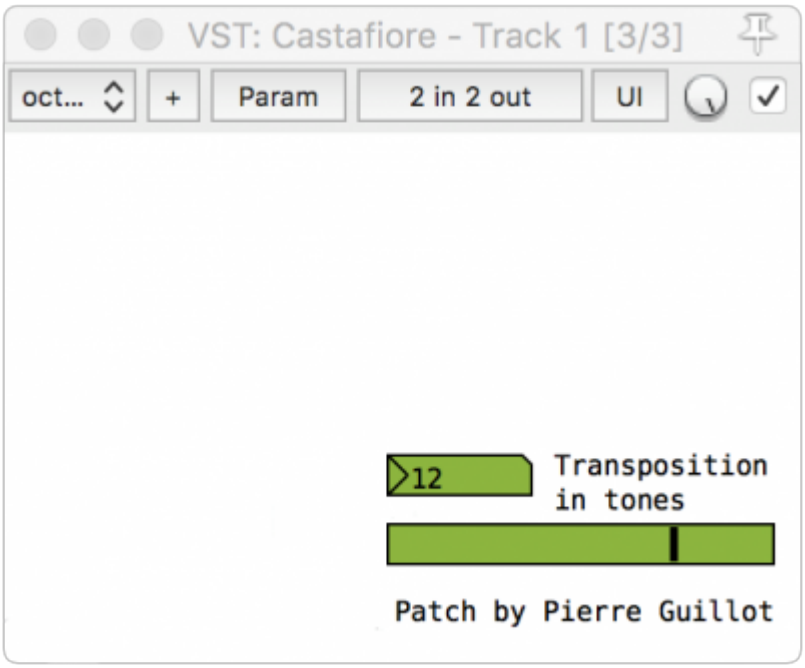
Figure 1. Création d?un nouveau *plugin*

Note : Fonctionnement de la création d?un nouveau *plugin* à partir d?un patch Pure Data et du *plugin* Camomile

Source : Pierre Guillot, 2018

Cette approche offre à la fois les avantages du *plugin*, notamment *via* la variété des stations de travail audionumérique à laquelle elle donne accès, ainsi que ceux de Pure Data, à savoir son dynamisme et sa modularité. Pour l?utilisateur, il s?agira de comprendre comment créer des patchs afin d?avoir accès à toutes les fonctionnalités offertes par les *plugins* et les stations de travail. Ces questions sont abordées dans la documentation fournie avec l?outil Camomile et ont été présentées à la Linux Audio Conférence en 2018 (Guillot, 2018). En parallèle, il est intéressant de revenir sur les raisons de la mise en ?uvre de cet outil, ainsi que sur les choix opérés lors de celle-ci. En effet, c?est de cela même que résultent les modalités d?utilisation de l?outil. Cet article revient donc dans la section 1 sur le contexte et les enjeux de cette mise en ?uvre, afin de présenter les spécifications techniques fonctionnelles qui ont permis de déterminer les choix de développement. Puis, dans la section 2, les travaux à la fois préexistants et parallèles au projet Camomile sont présentés, afin de mettre en avant les questions et les problématiques importantes qu?il est nécessaire de résoudre. À la suite de cela, dans la section 3, les axes importants de la mise en ?uvre de Camomile sont explorés, en allant des premiers développements en mai 2015 à la dernière version publiée en décembre 2017 (Figure 2). Enfin dans la section 4, ce travail est mis en perspective en revenant sur les questions qu?il reste à explorer mais aussi sur le potentiel offert par un tel outil.

Figure 2. Dernière version de Camomile



Note : Le *plugin* Castafiore, distribué en exemple avec la dernière version de Camomile et permettant de modifier la hauteur du signal sonore en fonction d?un paramètre de transposition contrôlable *via* l?interface graphique avec soit une boîte nombre, soit une glissière (*slider*) horizontale

Source : *plugin* Camomile et *plugin* Castafiore (crédits image : Pierre Guillot, 2020)

1. Contextes et enjeux

Le *plugin* Camomile a été élaboré afin de répondre à l?envie et la nécessité d?utiliser des patchs Pure Data au sein d?une station de travail audionumérique. Bien que relativement spécifique, ce besoin se manifeste dans des pratiques, des usages et des contextes variés, qui répondent à différents enjeux et desquels découlent des spécifications qui peuvent être complémentaires. Nous revenons ici sur ces questions, afin de comprendre et mettre en perspective les travaux préexistants présentés par la suite, ainsi que les choix qui ont été opérés lors de la mise en ?uvre de Camomile.

1.1. Contextes du projet

En 2015, au début du projet, la nécessité d?utiliser des patchs dans une station de travail audionumérique était liée au besoin de faciliter le contrôle des traitements audionumériques mis en ?uvre dans un contexte de temps différé. L?objectif est notamment de tirer profit de la ligne temporelle généralement présente dans les stations de travail audionumérique (6). De nombreux praticiens de l?informatique musicale ont d?ailleurs fait part de leurs attentes à ce sujet, ce qui amène notamment en 2013 l?équipe du CICM, dans le cadre des projets HOA (Se?des *et al.*, 2014), à concevoir un *plugin* VST offrant alors la possibilité de spatialiser des sources directionnelles en ambisonie (7) (Guillot *et al.*, 2013). Il a alors été envisagé de proposer d?autres traitements originaux de l?espace et du son ? tels que ceux offerts par les versions pour les logiciels Max (8) et Pure Data de la bibliothèque HOA (9) ? mais, faute de temps et en raison de la complexité des opérations nécessaires à leur mise en ?uvre, cet objectif n?a jamais pu être réalisé. À la suite du prolongement des expérimentations et à l?approfondissement des travaux de recherches sur la spatialisation du son, notamment *via* le logiciel Pure Data (Guillot, 2017), la seule solution viable pour répondre à ce problème semble être un *plugin* flexible et dynamique permettant de charger des patchs. Cette approche permet d?assurer une bonne reproduction du rendu sonore des traitements réalisés sous forme de patch au sein des stations de travail audionumérique, tout en limitant le temps de développement. Les logiciels de type *patcher* sont idéals pour l?expérimentation, mais prendre en compte et intégrer dans un *plugin* les nombreux et fréquents changements de ces expérimentations est très chronophage. Ainsi, l?enjeu du projet est aussi d?offrir un outil plus adapté à un contexte de développement et recherche. Enfin, la nécessité d?un tel outil s?est fait ressentir dans un cadre pédagogique. Le département de musique de l?université Paris 8 propose un cours de 2e année de licence d?Introduction à la programmation avec Max et Pure Data (10) où beaucoup d?étudiants découvrent pour la première fois les logiciels de type *patcher* (Puckette, 1988). L?un des défis pédagogiques est de démontrer le potentiel de ces outils *via*, entre autres, des conditions et des contextes concrets d?utilisations. Les étudiants sont familiers des stations de travail audionumérique et les utilisent généralement déjà fréquemment. Proposer un *plugin* permettant d?utiliser un traitement créé sous forme de patch directement dans un de ces logiciels offre une réponse à ce problème (11). Le projet Camomile tente donc de répondre à des problématiques provenant d?une approche d?utilisateurs, de développeurs, mais aussi d?enseignants.

1.2. Spécifications

Ces observations amènent à définir un certain nombre de spécifications fonctionnelles et techniques.

1.2.1. Intégration du moteur *patcher*

Afin de répondre à l?ensemble des usages dans des contextes compositionnels, expérimentaux, pédagogiques ou de développements, l?idéal serait évidemment que l?outil fonctionne sur le plus grand nombre de plateformes logicielles et supporte le plus grand nombre de formats de *plugin* audionumérique et de systèmes d?exploitation. L?objectif serait de concevoir un outil compatible avec les systèmes Linux, macOS et Windows (12), sous forme de *plugin* intégrant les technologies VST, Audio Unit ou encore LV2 (13). Sur un asp

1.2.2. Gestion du moteur *patcher* et du *plugin*

Évidemment, afin d?être utilisable, l?outil doit aussi proposer à l?utilisateur un moyen de définir et contrôler l?ensemble des fonctionnalités génériques d?un *plugin*, telles que les paramètres, les préréglages, la latence, etc. La possibilité, pour l?utilisateur souhaitant créer un patch destiné au *plugin* audio, de coder uniquement avec les langages initiaux de type *patcher* ? ou du moins de ne pas avoir à procéder à des opérations qui demandent des notions complexes de développement informatique, telles que la compilation d?un code ? pourrait constituer l?une des spécifications à ce sujet. En effet, si l?outil final est destiné à des étudiants, ce type d?approche peut rapidement décourager son utilisation. Cette spécification liée aux questions d?accessibilité est également importante dans un contexte de développement, car elle assure la portabilité d?un même patch sur l?ensemble des plateformes logicielles et des systèmes d?exploitation supportés par le *plugin*. Les logiciels de type *patcher* étant des applications de programmation graphique, il serait aussi intéressant de pouvoir offrir la possibilité de créer l?interface graphique utilisateur du *plugin via* l?interface originale du programme. Le deuxième enjeu principal de la conception de l?outil est de créer un système commun de gestion et de communication entre le logiciel de

type *patcher* et le *plugin*, et cela, afin de pouvoir définir les propriétés du *plugin* à partir du *patcher* et d?adapter le fonctionnement des patchs en fonction des informations envoyées par le *plugin*.

2. État de l?art

Avant de proposer une réponse à ces enjeux et ces spécifications en exposant les choix réalisés lors de la mise en ?uvre du *plugin* Camomile, nous rappelons les propositions les propositions préexistantes ou développées parallèlement, afin de s?en inspirer et d?éviter les problèmes que les créateurs ont pu rencontrer.

2.1. Des origines de Max à Max for Live

L?idée d?utiliser des patchs en tant que module d?extension au sein d?un système tiers n?est pas nouvelle. Dès 1988, l?un des objectifs du logiciel Max est notamment de pouvoir créer des synthétiseurs audionumériques sous forme de patch pouvant être utilisés sous forme de carte de traitement du signal (16) . Cette proposition ne restera longtemps qu?à l?état d?idée. En effet, à ses débuts, le logiciel était exclusivement orienté sur le traitement de messages et l?intégration du traitement du signal audionumérique dans Max n?apparaîtra qu?en 1991 (Puckette, 1991). C?est à la suite du développement de l?audionumérique, avec l?émergence des stations de travail audionumérique telles que Cubase en 1996 et du format VST (17), que l?idée pourra voir le jour, notamment en remplaçant les cartes de traitement du signal par des *plugins* audionumériques. En 1998, la société Cycling?74 propose une extension à Max nommée Pluggo (Zicarelli, 2002), permettant de charger des patchs au sein d?une station de travail audionumérique *via* un *plugin* VST. Cet outil offre nombre de fonctionnalités très complètes, telles que la création du moteur sonore, la création d?une interface graphique originale et d?un système permettant de configurer et gérer les paramètres, notamment à l?aide d?objets Max dédiés spécifiquement à cet usage. Le *plugin* de Pluggo fonctionne avec une version d?exécution de Max embarquée (18) qui permet de charger dynamiquement des patchs. Cet outil offre aussi la possibilité de générer de nouveaux *plugins* en embarquant directement les patchs dans des copies indépendantes et distinctes du *plugin* original de la distribution. Le développement du projet prend fin en 2006 avec la version 3.6.1 pour Max 4.6.2, ce qui la rend aujourd?hui dépréciée et restreint fortement toute utilisation. Une technologie équivalente est cependant réintégrée en 2009 *via* l?extension Max for Live, permettant de charger des patchs au sein du logiciel Ableton Live (19) et dont le fonctionnement est relativement similaire (20). Cet outil puissant, par ses nombreuses années de développement et d?usage, souffre néanmoins d?une double restriction. Il n?est réservé qu?à une seule station de travail audionumérique, Ableton Live, dont la technologie est fermée, tout comme celle du logiciel Max. Aussi, cela freine son appropriation dans un contexte de recherche, où les notions de propriété et d?ouverture sont importantes (21). Il en va de même dans un contexte d?utilisation lié à un cadre pédagogique, où le prix cumulé des technologies devient rapidement un frein à l?appropriation des outils par les étudiants universitaires.

2.2. De Pure Data au moteur libpd

Le logiciel Pure Data, par sa gratuité, son fonctionnement multiplateforme et son code source ouvert, est accessible à tout un chacun et accepte les ajouts et modifications par des personnes extérieures (22). Il est ainsi une excellente alternative à l?usage de Max et offre un terrain pour l?émergence d?outils adaptés aux besoins pédagogiques mais aussi de recherche. C?est donc naturellement vers cet outil que les développeurs se tournent lorsqu?il s?agit de s?approprier un moteur audio offrant un modèle de type *patcher*. Pure Data est déjà utilisé par de nombreuses plateformes et de nombreux outils logiciels en tant que moteur audio. Un des premiers projets utilisant cette technologie est PDa (Pure Data Anywhere), réalisé en 2003 par Günter Geiger, (Geiger, 2003) et visant à faire fonctionner le moteur audio du logiciel sur un PocketPC. Par la suite, en 2010, un ensemble de développeurs (23) proposent libpd, une bibliothèque en C embarquant le moteur (24) de Pure Data et offrant des interfaces de programmation en C++, Java, Processing, Objective-C et Python, qui permettent notamment un support pour les plateformes Android and iOS (Wilcox, 2016). Cette bibliothèque est utilisée dans de très nombreux projets (25), dont les applications pour téléphones et tablettes tactiles iOS (26) avec PdParty de Dan Wilcox (*Id.*) ou son homologue pour Android développée par Chris McCormick, PdDroidParty (27), dont elle est inspirée. Dès lors qu?il est possible d?embarquer le moteur de Pure Data au sein d?autres applications *via* libpd, plusieurs projets de *plugins* permettant de charger des patchs Pure Data voient le jour. Par exemple, le *plugin* au format LV2, PdLV2 (28), qui fonctionne sur les systèmes d?exploitation Linux et dont le

développement date d?au moins 2013, ou plus récemment le *plugin* VST PdPulp [(29)] qui fonctionne sur le système d?exploitation macOS et dont le développement est contemporain de celui de Camomile. Ces projets sont extrêmement expérimentaux et possèdent sensiblement moins de fonctionnalités qu?un outil tel que Max for Live. Ils sont par conséquent encore difficilement utilisables dans un contexte de production. Il n?est donc pas nécessaire de revenir avec précision sur ces mises en ?uvre, mais il est important de relever un problème crucial lié directement à l?architecture du moteur Pure Data. Le logiciel a été pensé comme une application autonome et le résultat concret pour les utilisateurs et les développeurs est l?unique instance d?un *plugin* pouvant être chargée dans la station de travail audionumérique [(30)]. La résolution de ce problème est primordiale car l?utilisation de multiples instances d?un même *plugin* dans les chaînes de traitement audio est ancrée dans les usages.

2.3. Au-delà de libpd

Deux approches ont néanmoins réussi, si ce n?est à résoudre le problème, du moins à le contourner. L?une se trouve au sein du *plugin* PdVst~ originellement développé en 2004 par Joseph Jarlo [(31)] puis repris depuis 2016 par Jean Yves Gratius [(32)]. Dans ce *plugin* VST, Pure Data n?est pas réellement embarqué. Pour chaque instance du *plugin*, une instance de l?application de Pure Data est démarrée de telle sorte que chaque application possède un espace mémoire qui lui est propre, évitant dès lors des conflits [(33)]. Aussi, le *plugin* fonctionne en réalité comme un pont entre ces applications Pure Data associées aux *plugins* et la station de travail audionumérique. Le problème majeur reste que cet outil est restreint au système d?exploitation Windows. Il serait donc nécessaire d?étudier son potentiel fonctionnement sur d?autres systèmes d?exploitation, mais aussi de vérifier si cette approche n?implique pas de latence et si la communication entre les instances de Pure Data et la station de travail audionumérique *via* ce système est réellement stable, notamment dans un contexte de *multithread* [(34)]. Une deuxième approche est proposée par la plateforme Heavy [(35)] distribuée par la société Enzien Audio, qui offre la possibilité de générer directement un *plugin* VST à partir d?un patch Pure Data. La plateforme offre un système de compilation en ligne, mais au lieu d?intégrer Pure Data au *plugin*, Heavy interprète le patch avec son propre moteur qui, quant à lui, offre un fonctionnement multi-instance. La force première de cette approche est aussi son principal défaut. En effet, la plateforme n?utilisant pas réellement Pure Data, elle ne propose qu?un sous ensemble de ses fonctionnalités, ce qui restreint les possibilités et implique immanquablement un retard par rapport aux mises à jour de la distribution de Pure Data Vanilla [(36)]. De plus, l?exacte similitude du rendu sonore entre le patch dans le logiciel Pure Data et le *plugin* généré n?est pas assurée, du fait de la réécriture totale ou partielle du moteur, afin de le rendre opérable dans un système multi-instance.

3. Mise en ?uvre de Camomile

Lors de la définition des spécifications techniques et fonctionnelles de l?outil Camomile, deux enjeux majeurs sont apparus. Nous exposons donc ici les choix opérés lors de la mise en ?uvre, d?une part, au niveau de l?intégration du moteur *patcher* dans le *plugin* et, d?autre part, au niveau de la création d?un système de communication entre le moteur *patcher* et le *plugin*. Ces choix sont aussi confrontés à l?état de l?art présenté dans la partie précédente. À ce jour, deux versions majeures de Camomile ont été réalisées. Le changement de version correspond à une amélioration de la technologie utilisée. Sans pour autant revenir sur l?ensemble des spécificités de chaque version, les avantages et les conséquences de ces choix seront discutés.

3.1. Intégration du moteur dans le *plugin*

Pour les raisons évoquées précédemment, le *plugin* vise à utiliser le moteur de Pure Data qui offre une licence libre [(37)] et un usage gratuit, tout en ayant une large communauté d?utilisateurs très actifs (Puckette, 2004). Par ailleurs, l?interface de programmation applicative Juce [(38)] a été choisie, car elle offre entre autres la possibilité, avec un seul code générique, de créer des *plugins* pour les formats LV2 [(39)], VST et Audio Unit, ainsi que pour les systèmes d?exploitation Windows, Linux et macOs. L?enjeu est alors de réussir à embarquer le moteur de Pure Data au sein du *plugin*. Or, comme cela a été évoqué précédemment, la difficulté première est de rendre le moteur de Pure Data compatible avec un usage multi-instance dans un contexte de *multithread*.

3.1.1. L?approche séquentielle

Les développements de Pure Data pour la version 0.46, avec le début de l?intégration par Miller Puckette de la gestion du multi-instance, ont simplifié la première mise en ?uvre de Camomile. Ces modifications du c?ur du moteur, bien qu?insuffisantes à une mise en ?uvre directe dans le contexte d?un *plugin* audio, ont permis de proposer une première stratégie consistant à forcer le caractère séquentiel des opérations. Pour cela, une interface similaire à libpd a été créée pour répondre spécifiquement à ce problème. La méthode consiste à restreindre l?accès au moteur à une instance à la fois, tout en limitant chaque instance à une seule opération, en utilisant un système de verrous (*locks*). Afin d?offrir un accès dans un contexte de *multithread*, le système utilise des listes chaînées pour enregistrer les messages et les instructions qui sont lues et exécutées par la suite de manière séquentielle, avant de traiter le signal audionumérique de chaque instance. Le second enjeu est de simplement faire concorder l?interface Juce avec l?interface enveloppant le moteur Pure Data. En avril 2016, une première version fonctionnelle destinée à un large usage (40) a pu être publiée. Elle a été testée sur de nombreuses plateformes logicielles et à chaque fois sur les systèmes d?exploitation pour lesquels elles sont disponibles (41). Malgré l?approche radicale consistant à forcer le caractère séquentiel des opérations, le système était jouable et n?offrait pas d?artefacts en dehors d?une hausse de l?utilisation du processeur ? résultat de l?attente de déverrouillage d?une instance par une autre.

3.1.2. L?approche parallèle

Par la suite, avec la version 0.47 de Pure Data, les derniers problèmes de l?utilisation multi-instance dans un contexte de *multithread* ont été résolus en utilisant notamment la mémoire locale de *thread*. En gardant la mémoire associée à une instance locale sur chaque *thread*, cela permet d?utiliser en concurrence plusieurs instances sur des *threads* différents (42). Néanmoins, l?usage d?une telle approche demande de revoir une grande partie de l?interface mise en ?uvre précédemment. Dès lors, afin que ce travail puisse être utile à une plus large communauté de programmeurs (Brinkmann *et al.*, 2011), le choix a été fait d?utiliser libpd et d?y intégrer ces améliorations. Les interfaces de programmation élémentaires de libpd pour le multi-instance ont été mises à jour, notamment par Miller Puckette et Dan Wilcox. L?enjeu était alors de rendre certains aspects importants ? notamment la réception des messages, des événements MIDI ou des notifications de console ? compatibles avec le multi-instance. Aussi, la mise à jour pour ce dernier et le *multithread* du noyau de Pure Data n?a pas encore été réellement confrontée à un usage concret. Cette mise en ?uvre était l?occasion de tester l?approche et de corriger les quelques cas difficiles à prévoir (43). En décembre 2017, la deuxième version de Camomile a pu être publiée et grâce au relatif succès de la précédente version, de nombreux retours d?utilisation constructifs ayant permis de déployer rapidement les outils sur les différents systèmes d?exploitation (44).

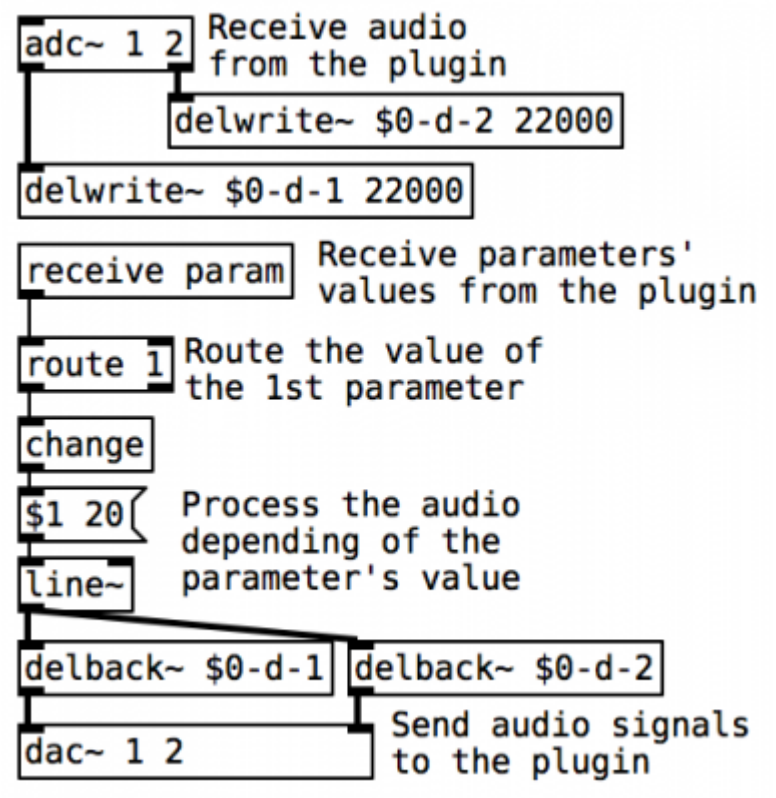
3.2. Communication entre le moteur et le *plugin*

La gestion du moteur *patcher* par le *plugin* et réciproquement des caractéristiques du *plugin via* le moteur recouvre plusieurs aspects.

3.2.1. La communication et le partage de données

Un premier aspect important de la gestion du moteur par le *plugin* concerne la communication et, de manière générale, le partage de données audio, MIDI ou encore de type message. La réception et l?envoi du signal audionumérique du *plugin* vers le moteur *patcher* et réciproquement peuvent être réalisés respectivement avec les objets **adc~** et **dac~**. De manière analogue, les événements MIDI reçus par le *plugin* peuvent être récupérés dans un patch *via* les objets MIDI natifs de Pure Data. Des événements MIDI peuvent également être générés par le patch et envoyés vers le *plugin via* les objets MIDI équivalents pour la sortie. Enfin, la réception de messages du *plugin* vers un patch est quant à elle réalisée avec les objets **receive** et les messages sont transmis du patch vers le *plugin* avec les objets **send**. Ces messages sont notamment utilisés pour modifier dans un patch les valeurs des paramètres, mais aussi afin d?être notifié de leurs changements par la station de travail audionumérique (45) (Figure 3).

Figure 3. Patch Pure Data



Note : Patch Pure Data utilisé dans le *plugin* Bulgroz où le signal en entrée, reçu par l?objet **adc~**, est traité en fonction de la valeur du premier paramètre, reçu par l?objet **receive**, avant d?être renvoyé au *plugin via* l?objet **dac~**

Source : *plugin* Bulgroz (crédits image : Pierre Guillot, 2018)

L?avantage de cette approche est que, contrairement à Max for Live, aucun objet externe et spécifique à Camomile n?est nécessaire au fonctionnement d?un patch dans le *plugin*. Cela évite une possible obsolescence des patchs et les rend utilisables en dehors du *plugin*. Il est intéressant de remarquer qu?afin d?assurer une bonne distribution des messages entre les instances, la première version nécessitait l?utilisation des indices spécifiques aux patchs dans les symboles de réception et d?envoi [(46)]. La deuxième version, avec sa gestion du multi-instance plus poussée, permet de se libérer de cette contrainte.

3.2.2. Le chargement des patchs

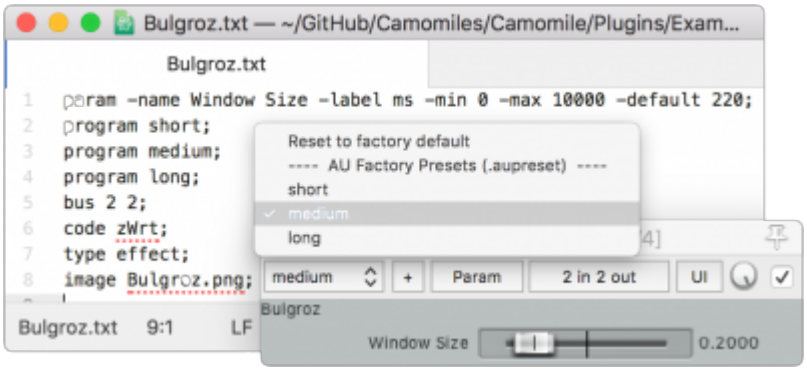
Un deuxième aspect notable de la gestion du moteur par le *plugin* concerne la méthode de chargement des patchs. Deux approches peuvent être envisagées et ont été mises en ?uvre respectivement dans la première et la deuxième version du *plugin*. La première version offrait la possibilité de charger des patchs dynamiquement *via* une boîte de dialogue. Cette approche possède l?avantage d?ouvrir rapidement un patch dans la station de travail audionumérique. Associée avec la fonction de rechargement du patch courant, c?est une fonctionnalité très utile [(47)] pour la création des patchs, mais qui pose néanmoins des problèmes. Plusieurs aspects du fonctionnement du *plugin* dépendent du patch chargé, comme les paramètres, les préréglages, les configurations de canaux supportées, etc. Or, changer de patch après le chargement initial du *plugin* par la station de travail audionumérique implique de modifier ces éléments à la volée, ce qui n?est pas supporté par de nombreuses stations de travail audionumériques [(48)]. De plus, la localisation des patchs n?est pas assurée lorsque les projets sont partagés entre les utilisateurs [(49)]. Enfin, utiliser l?interface graphique pour le chargement du patch empêche l?utilisation du *plugin* dans les stations de travail qui ne supportent justement pas les interfaces graphiques spécifiques aux *plugins* [(50)]. Pour répondre à ce problème, la deuxième version de Camomile a choisi d?associer chaque patch ? ou un ensemble de patchs ? à une nouvelle copie du *plugin* Camomile original. En liant et en intégrant de la sorte les patchs au fichier binaire du *plugin*, le chemin relatif entre les deux est toujours préservé. À la première instanciation du *plugin* dans la station de travail, le ou les patchs associés sont automatiquement chargés et l?ensemble des informations nécessaires au fonctionnement du *plugin* ? paramètres,

préréglages, configurations des canaux, etc. ? peut être récupéré. Par la suite, le patch d?une instance du *plugin* peut être rechargé à la volée [(51)] ? permettant donc de modifier le patch, que ce soit le moteur sonore ou l?interface graphique ?, mais les informations fournies à la station de travail audionumérique, et qui doivent nécessairement rester inchangées, ne sont pas modifiées.

3.2.3. Définition des propriétés du *plugin*

Finalement, il est important d?examiner la manière de définir ces différentes informations liées au fonctionnement du *plugin*. Là encore, l?approche est différente entre les deux versions de Camomile. La première version se limitait aux seules informations des paramètres. Aussi, il avait semblé judicieux de définir ces derniers à l?aide des interfaces graphiques utilisateur du patch principal. Par exemple, une glissière (*slider*) ou une boîte nombre générait automatiquement un paramètre en fonction du label et de la plage de valeur jouable définie dans les propriétés de l?objet graphique [(52)]. Cette approche possède l?avantage de savoir dans le *plugin* quand la valeur d?un paramètre est modifiée par l?utilisateur *via* cette interface et de simplifier la mise en ?uvre du *plugin* et la création du patch. Cependant, un tel choix possède des inconvénients majeurs. L?approche se complexifie dès lors que l?utilisateur souhaite recourir à plusieurs interfaces graphiques pour contrôler un même paramètre (Figure 2), mais surtout, l?ordre des paramètres [(53)] dépend de l?ordre de création des interfaces graphiques. Aussi, dans le processus de création d?un patch pour le *plugin*, définir l?ordre des paramètres peut nécessiter de recréer les interfaces graphiques et leurs connexions. Cela peut être laborieux et source d?erreurs car ce n?est malheureusement pas explicite. Enfin, il restait à définir de nouvelles méthodes pour configurer les autres options du *plugin*, telles que les préréglages, les configurations audio et MIDI supportées, etc.

Figure 4. *Plugin* Bulgroz



Note : En arrière-plan, le fichier texte permettant de définir les propriétés du *plugin* Bulgroz avec Camomile, dont notamment le paramètre de taille de fenêtre et trois préréglages. Au premier-plan, l?interface graphique par défaut du *plugin* dans le logiciel Reaper permet de modifier la valeur du paramètre et de sélectionner les préréglages.

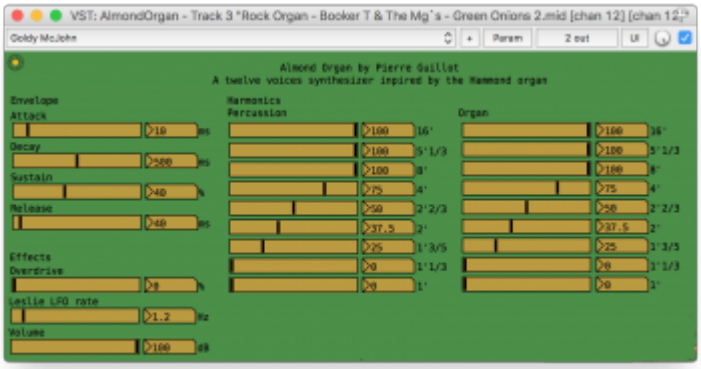
Source : *plugin* Camomile, *plugin* Bulgroz et logiciel Reaper (Cockos, 2006) (crédits image : Pierre Guillot, 2018)

Afin de remédier à ces problèmes, la dernière version de Camomile utilise un fichier texte additionnel pour définir l?ensemble des informations nécessaires à un *plugin*. La syntaxe de ce fichier est simple. Chaque ligne permet de définir une option, de rajouter un paramètre ou un préréglage [(54)], ce qui ouvre la possibilité d?écrire ou de charger le fichier *via* l?objet **text** de Pure Data (Figure 4). Enfin, un système de notifications du *plugin* vers le patch et réciproquement du patch vers le *plugin* permet d?associer les interfaces graphiques aux paramètres [(55)]. La version actuelle du *plugin* permet donc un support complet de l?ensemble des fonctionnalités génériques et des spécifications des *plugins*, tout en préservant une cohérence avec le langage et les usages de Pure Data.

4. Bilan et perspectives

Camomile offre à présent un outil compatible sur les trois principaux systèmes d?exploitation grand public avec la plupart des stations de travail audionumérique supportant des *plugins*. Les choix opérés dans la dernière version du *plugin* ? que ce soit pour le chargement de patchs, la communication entre le moteur et le *plugin* ou encore la création de patchs, ainsi que la définition des propriétés du *plugin* ? ont permis de supporter pleinement l?ensemble des fonctionnalités usuellement offert par ce type de formats : la gestion des paramètres, des préréglages, des configurations audio, des événements MIDI, des informations de la tête de lecture [(56)] ou encore la création d?une interface graphique adaptée et fonctionnelle. Ainsi, des patchs correctement configurés et tirant parti de l?ensemble de ces fonctionnalités peuvent permettre de générer des *plugins* rivalisant, sur l?aspect sonore ou l?aspect ergonomique, avec des *plugins* codés et compilés de manière habituelle. De plus, l?outil n?est pas seulement limité à un usage expérimental, mais peut très bien être utilisé dans un contexte de production musicale abouti ou dans un contexte pédagogique [(57)] (Figure 5).

Figure 5. Le *plugin* AlmondOrgan au format VST



Note : Le *plugin* AlmondOrgan au format VST, un synthétiseur douze voix inspiré par l?orgue Hammond, créé avec Camomile et intégrant de multiples fonctionnalités comme la gestion des événements MIDI, des paramètres et des préréglages

Source : *plugin* AlmondOrgan (crédits image : Pierre Guillot, 2018)

Camomile est, en outre, un outil extrêmement intéressant pour les développeurs car il permet de créer rapidement des prototypes de *plugins* en prenant en compte tous les aspects de la mise en ?uvre : moteur audio, interfaces graphiques, messages, etc. Au-delà des prototypes, les *plugins* créés avec Camomile peuvent être publiés en tant que tels. Le *plugin* jouit déjà d?une relative notoriété dans la communauté d?utilisateurs de Pure Data et de l?audionumérique libre de manière générale [(58)], mais il serait intéressant d?ouvrir la communauté d?utilisateurs *via* le partage direct de *plugins* créés avec Camomile. Cela pourrait donner envie à des utilisateurs exclusifs des stations de travail audionumérique de s?intéresser aux approches de type *patcher*. Sur le plan du développement, Camomile aura été un excellent terrain d?expérimentation et de mise à l?épreuve du support *multithread* et multi-instance de Pure Data. Il aura permis de révéler un certain nombre de problèmes, dont de nombreux ont été corrigés. Enfin, certains points sont néanmoins encore à explorer et d?autres problèmes restent à corriger. C?est le cas notamment de la gestion des bibliothèques externes d?objets qui nécessitent *a priori* d?être compilées avec le *plugin*. Il existe deux raisons à cela. D?une part, les stations de travail audionumériques semblent pour la plupart bloquer le chargement dynamique de bibliothèques externes. D?autre part, il est nécessaire de vérifier si les bibliothèques externes n?interfèrent pas avec les conditions d?activation du support du multi-instance et du *multithread* de Pure Data [(59)]. Il serait aussi intéressant de réfléchir à une amélioration ou une extension de la création et de la gestion de l?interface graphique. Il s?agirait de prendre en compte le système de structure de données graphiques (Puckette, 2002) ou d?offrir un système dynamique permettant d?intégrer ses propres graphismes aux interfaces, afin d?augmenter les possibilités [(60)]. Enfin, il serait intéressant d?intégrer la génération de *plugins* au format Audio Unit v3 [(61)], qui permettrait de charger les *plugins* sur des tablettes et téléphones iOS, mais aussi pour Unity [(62)].

Remerciements

Je tiens à remercier toute la communauté de développeurs de Pure Data et de libpd, notamment Miller Puckette et Dan Wilcox, pour leurs conseils et leurs explications, mais aussi les utilisateurs de Camomile pour leurs retours d?utilisation et leurs suggestions. Je tiens aussi à remercier toute l?équipe du CICM pour l?intérêt porté à ce projet et particulièrement Alain Bonardi et Eliott Paris pour leurs remarques et leurs suggestions.

---

1. Les différentes versions du *plugin* sont disponibles en ligne sur le répertoire Github du projet. Le code source est ouvert, libre et gratuit et est disponible sur ce même répertoire. Depuis la version 0.1.0, les sources sont distribuées sous licence GNU GPLv3. Les sources des versions antérieures sont distribuées sous licence BSD 3. https://github.com/pierreguillot/Camomile/releases [consulté en janvier 2018]

2. Pure Data est un logiciel de type *patcher* libre et gratuit, créé par Miller Puckette à l?université de San Diego en Californie. http://msp.ucsd.edu/software.html [consulté en janvier 2018]

3. Le format LV2 développé par David Robillard est le successeur du format LADPSA. http://lv2plug.in/ns/ [consulté en mars 2020]

4. Les formats de *plugin* audionumérique VST (*virtual studio technology*) 2 et 3 sont développés par la société Steinberg. https://www.steinberg.net/en/company/developers.html [consulté en janvier 2018]

5. Le format de *plugin* audionumérique Audio Unit est développé par Apple. https://developer.apple.com/audio/ [consulté en janvier 2018]

6. Dans cet article, il ne s?agit pas d?approfondir la question de la ou des représentations du temps offertes de manière native au sein du logiciel Pure Data, d?autant qu?il est possible de combiner Pure Data à d?autres outils d?écriture temporelle tels que les logiciels ossia développé au LaBRI (De La Hogue *et al.*, 2014) ou Iannix (Coduys, Lefèvre et Pape, 2003). Il s?agit simplement de mettre en avant un besoin et une attente qui est plutôt liée à des questions d?accessibilité et de facilité de mise en ?uvre et d?utilisation.

7. Le *plugin* est à présent non fonctionnel, mais il est cependant toujours disponible sur le site du projet HOA. http://www.mshparisnord.fr/hoalibrary [consulté en janvier 2018]

8. Le logiciel Max (Favreau *et al.*, 1986), originellement développé à l?Ircam par Miller Puckette, est aujourd?hui développé et distribué par la société Cycling?74. https://cycling74.com/ [consulté en janvier 2018]

9. Ces opérations sont par exemple construites autour de la synthèse granulaire ou d?un filtre à réponse impulsionnelle infinie réalisés sous forme de patch dans les logiciels Max et Pure Data. Leur complexité réside alors dans la recréation en C et C++ de l?ensemble des traitements audionumériques des objets utilisés, ainsi que leurs réseaux de connexions, mais aussi les interfaces graphiques utilisateurs, même basiques, permettant de contrôler ces traitements.

10. Ce cours a été dispensé depuis de nombreuses années par Anne Sèdes, Alain Bonardi, puis par moi-même en 2016 et 2017 et à présent par Eric Maestri qui intègre notamment le logiciel Kiwi (Paris *et al.*, 2017) à l?apprentissage.

11. Au cours de ces années d?enseignement, savoir s?ils pouvaient utiliser leurs patchs sur des stations de travail audionumérique était en effet une question récurrente des étudiants.

12. Il est raisonnable d?affirmer que l?usage de cet outil sur d?autres systèmes d?exploitation ou avec d?autres formats de *plugin* audionumérique se révèle purement anecdotique pour le moment.

13. Les informations relatives au format LV2 sont disponibles sur le site internet. http://lv2plug.in/ [consulté en janvier 2018]

14. Cette spécification est d?autant plus en accord avec une approche de recherche et recoupe des questions pédagogiques, car les applications libres et ouvertes sont très souvent gratuites ou du moins, plus accessibles.

15. Système permettant l?exécution de plusieurs tâches en parallèle, ce qui est souvent le cas sur les ordinateurs modernes et les logiciels actuels.

16. « Max a des crochets pour patcher un synthétiseur numérique, qui peut être (comme maintenant) un programme d?interprétation sous Macintosh ou peut (à l?avenir) être une carte de traitement de signal enfichable ». Je traduis « Max has hooks for patching a digital synthesizer, which may just be (as now) an interpreting program in the Macintosh or may (in the future) be a plug-in signal-processing card. » (Puckette, 1988, p. 420).

17. En 1996, la société Steinberg publie la station de travail audionumérique Cubase 3.02, avec notamment l?intégration de la technologie VST ; d?après le site ArtisteAudio. https://artisteaudio.fr/steinberg-cubase/ [consulté en janvier 2018]

18. Il n?est possible que de jouer le patch. L?édition doit être réalisée au sein du logiciel Max original.

19. Le transfert de la technologie de Pluggo vers Max for Live a été annoncé en mai 2009 sur le site de Cyling?74. https://cycling74.com/newsletters/pluggo-technology-moves-to-max-for-live [consulté en janvier 2018]

20. Évidemment, de nouvelles fonctionnalités sont apparues et/ou ont été modifiées.

21. La comparaison entre le développement de Pure Data et de jMax est un bon exemple de ce problème (Puckette, 2004).

22. « Pd was instantly embraced by a huge, and extremely hip, community of users who have taken it far beyond my wildest dreams of it [...] One meaning of Pd was ?Public Domain? » (*Ibid.*, p. 200)

23. Les créateurs de libpd sont, à l?époque, une équipe rencontrée sur internet et composée de Peter Brinkmann, Peter Kirn, Richard Lawler, Chris McCormick, Martin Roth et Hans-Christopher Steiner.

24. Le moteur est ici pris dans son sens le plus large, comprenant la partie dédiée à l?audio, aux messages, au MIDI et autres normes, telles que l?OSC.

25. Il serait vain de vouloir citer l?ensemble de ces projets, surtout qu?une majeure partie semble rester fermée ou inaccessible.

26. Système d?exploitation mobile développé par Apple. https://www.apple.com/fr/ios/ios-14 [consulté en novembre 2020]

27. L?application est disponible sur le site PdDroidParty. http://www.droidparty.net/ [consulté en novembre 2020]

28. Le *plugin* est originellement développé par l?utilisateur de Github UnknownError et est publié depuis 2013 sur le répertoire Github (https://github.com/unknownError/pdLV2-stereo, consulté en janvier 2018). Depuis 2016, il est mis à jour et maintenu par Alex Norman et accessible sur son répertoire Github (https://github.com/x37v/pdlv2, consulté en janvier 2018).

29. Le *plugin* est développé par Karl Pannek and Oliver Greschke depuis juillet 2015 et est disponible sur le site https://github.com/logsol/Pd-Pulp [consulté en novembre 2020]

30. Ce diagnostic a aussi été fait lors des premiers essais de mise en ?uvre de Camomile. Cependant, les développements et mises à jour parallèles de Pure Data permettront, comme cela sera présenté dans la suite de l?article, de proposer une solution honorable et intermédiaire, avant d?arriver à une solution optimale avec les améliorations plus récentes de Pure Data.

31. https://puredata.info/downloads/pdvst [consulté en novembre 2020]

32. Le *plugin* et les informations relatives au projet sont disponibles *via* le répertoire Github de Jean Yves Gratius. https://github.com/jyg/pure-data [consulté en janvier 2018]

33. Cette approche possède aussi l?avantage éventuel, comparée aux approches présentées précédemment, de pouvoir utiliser directement l?interface graphique native. Il n?est néanmoins pas forcément souhaitable de vouloir utiliser l?interface graphique originale de Pure Data au sein d?un *plugin* audionumérique. Une interface spécifique et plus adaptée peut potentiellement être plus appropriée. Un autre avantage est la possibilité de charger les bibliothèques d?objets externes.

34. Faute de moyens techniques et de temps, cela n?a pas pu être réalisé pour le moment.

35. La plateforme Heavy a été disponible à partir 2016 mais le projet a été abandonné courant 2019.

36. Distribution principale de l?application réalisée par Miller Puckette.

37. Pure Data et le projet libpd sont tous deux sous la même licence définie par « Standard Improved BSD » et distribuée avec les codes sources.

38. Juce est une interface de programmation applicative orientée vers le traitement du signal audionumérique distribuée par la société Roli. https://juce.com/ [consulté en janvier 2018]

39. JUCE ne supporte pas nativement le format LV2, mais le travail de Filipe Coelho a permis d?intégrer le support de ce format dans JUCE.

40. La version 0.0.7 est toujours disponible sur le répertoire Github du projet.

41. Une liste non exhaustive de ces plateformes ? comprenant notamment les logiciels Reaper, Cubase, Ableton Live, Tracktion, Ardour, etc. ? et des systèmes d?exploitation ? Windows, macOS et Linux en 32/64bits ? est disponible sur le wiki du répertoire Github.

42. L?accès à une même instance *via* plusieurs *threads* peut néanmoins toujours causer des problèmes et l?accès sur un *thread* de plusieurs patchs est obligatoirement séquentielle.

43. Les dernières modifications devraient être acceptées prochainement sur les branches principales de libpd et Pure Data.

44. Cette version a déjà été testée sur un grand nombre de plateformes logicielles ? telles que Reaper, Ableton Live, Tracktion, Bitwig, etc. ? et de systèmes d?exploitation ? Windows 32 et 64bits, macOS 32 et 64bits et Linux 64bits.

45. Ces approches peuvent être aussi utilisées pour la gestion des préréglages, ou encore pour la configuration des canaux audio.

46. Ces indices peuvent être intégrés en utilisant les caractères *$0* dans ces symboles.

47. Avec la nouvelle version, il est néanmoins possible de retrouver un tel fonctionnement en chargeant dynamiquement des patchs au sein d?un patch principal. Cette opération a été réalisée dans le *plugin* PdStal donné en exemple de Camomile.

48. Cette approche est normalement interdite par la norme VST. Une restriction est définie dans la documentation du VST 3 Plug-In SDK. https://github.com/steinbergmedia/vst3sdk [consulté en janvier 2018]

49. Les patchs n?étant pas directement associés au projet en cours, au fichier binaire du *plugin* ou à la station de travail, son chemin est absolu et lorsqu?un projet est partagé sur un autre ordinateur, le *plugin* ne parvient pas à retrouver le patch. Il est donc nécessaire de redéfinir manuellement tous les chemins.

50. Certaines stations de travail audionumériques supportent encore mal les interfaces graphiques des *plugins*. C?est le cas notamment du logiciel libre et gratuit Audacity. http://audacity.fr/ [consulté en janvier 2018]

51. Cette fonctionnalité peut être très utile, notamment lors de la création du *plugin*. La version 1.0.4 du *plugin* offre, pour cela, la possibilité de recharger le patch manuellement ? en cliquant sur un bouton dédié ? ou automatiquement, à chaque nouvelle sauvegarde du patch.

52. Cette approche est aussi définie dans la documentation de la version 0.0.7 du *plugin* disponible sur le wiki du répertoire Github.

53. L?ordre des paramètres est très important, car les stations de travail audionumérique utilisent leurs indices (ou positions), et non leurs noms, comme identifiants des paramètres pour organiser les informations qui leurs sont relatives, telles que les automations.

54. Cette approche est définie dans la documentation du *plugin* et sera développée plus longuement dans une prochaine publication.

55. Cette approche est également définie dans la documentation du *plugin* et sera aussi développée plus longuement dans une prochaine publication.

56. Il est en effet possible d?utiliser toutes les informations fournies par la station de travail audionumérique, telles que le chiffrage de la mesure, le tempo de la mesure, la position de la tête de lecture, etc.

57. Le *plugin* a notamment été présenté à l?université Paris 8 aux étudiants en 2016 et 2017, dans le cadre des cours d?Introduction à la programmation avec Max et Pure Data 1 dispensés par moi-même et en 2018, aux étudiants du cours de Composition électroacoustique 2 dispensé par Alain Bonardi et suite auquel ils ont été invités à créer leurs propres *plugins*.

58. De nombreux *plugins* ont été créés avec Camomile ; certains d?entre eux sont consultables sur la page dédiée à cet outil du site (https://patchstorage.com/platform/camomile, consulté en avril 2020) et dans la rubrique d?exemples du projet (https://github.com/pierreguillot/Camomile/wiki/Demos-and-tutorials, consulté en avril 2020).

59. Plusieurs tentatives ont été réalisées sur le sujet mais aucune solution viable n?a encore été trouvée.

60. Le *plugin* offre déjà la possibilité de charger des images de fond (Figure 2 et Figure 5), mais il serait intéressant de pouvoir replacer les graphismes natifs de objets de Pure Data, tels que l?interrupteur (*toggle*), la glissière (*slider*) ou encore la boîte nombre (*numbox*), en utilisant des séries d?images données

par l?utilisateur.

61. Grâce au support de ce format par JUCE, il peut être déjà envisagé de publier une version Audio Unit v3 de Camomile, mais, faute de temps et de moyen techniques, il n?a pas encore été possible de tester l?approche.

62. Unity est moteur de jeu multiplateforme développé par Unity Technologies (https://unity.com, consulté en avril 2020). JUCE supporte le format de *plugin* de Unity. De premières tentatives d?intégration de Camomile ont déjà été réalisées, mais des verrous techniques doivent encore être résolus avant que l?outil soit pleinement fonctionnel sur cette plateforme logicielle.

Pour citer ce document:
Pierre Guillot, Â« Camomile, enjeux et développements d?un *plugin* audio embarquant Pure Data Â», *RFIM* [En ligne], Numéros, n° 7-8 - Culture du code, Mis Ã jour le 21/12/2020
URL: http://revues.mshparisnord.org/rfim/index.php?id=602