CECILIA 5, LA BOÎTE À OUTILS DU TRAITEMENT AUDIO-NUMÉRIQUE

Olivier Bélanger
iACT (institut Arts Cultures et Technologies)
Université de Montréal, Faculté de Musique

RÉSUMÉ

Le logiciel de traitement sonore Cecilia est une boîte à outils pour la manipulation du son, utilisée autant en pédagogie qu'en production musicale. Grâce à sa panoplie de modules de traitement originaux et à son interface graphique modulaire, Cecilia représente un environnement idéal pour l'exploration et la création sonore. La version 5 [2], dont le développement a débuté en octobre 2011, constitue une réécriture complète de l'infrastructure du logiciel. Une modernisation qui aura permis l'ajout de plusieurs nouvelles fonctionnalités au logiciel. Cet article se divise en deux sections. Dans un premier temps, seront illustrés les mécanismes que met en place le logiciel afin de favoriser une expérience de travail efficace et créative. Nous y retrouverons des outils tels que le grapheur, la lecture à boucle variable de fichiers son, la communication MIDI [1] et OSC [2] ainsi que la génération algorithmique de trajectoires de contrôle. Ensuite, sera détaillé l'API [3] de Cecilia, qui offre une interface simplifiée à l'utilisateur désirant développer de nouveaux modules, tout en bénéficiant de la flexibilité de l'interface graphique fournie par l'environnement. Cecilia 5 est un logiciel gratuit, de sources libres et fonctionnant sur toutes les plates-formes majeures.

1. INTRODUCTION

Cecilia fut initialement développé, de 1995 à 1998, par Jean Piché et Alexandre Burton [4] à la faculté de musique de l'Université de Montréal. L'objectif initial du projet était de pourvoir les studios de composition d'une plateforme numérique unifiée comme environnement de travail. La structure modulaire du logiciel permettait, à l'intérieur d'un environnement unique, d'effectuer une multitude de tâches reliées à la création de musiques concrètes. La version initiale de Cecilia fut écrite en TCL/TK, une combinaison constituée d'un langage de programmation interprété (TCL) et d'une bibliothèque graphique (TK), permettant le développement rapide d'applications multiplateformes. Le moteur audio utilisé pour effectuer les tâches relatives au traitement de signal était Csound [6], un langage de programmation pour la création sonore développé par Barry Vercoe au MIT. Jean Piché a poursuivi le développement de Cecilia, en conservant la structure originale, jusqu'en 2008. La version officielle était alors Cecilia 2.5. En 2009, afin de permettre l'ajout d'un certain nombre de fonctionnalités et une maintenance plus aisée du logiciel, un changement d'architecture s'opère et Cecilia 4 est reconstruit à neuf en remplaçant TCL par Python [5], un autre langage de programmation interprété, plus moderne, orienté-objet et soutenu par une communauté très active. L'interface de Cecilia est complètement redessinée avec la librairie graphique WxPython [3] et Csound demeure le langage assurant les services audio de l'application. Cette version sera maintenue et développée, par Olivier Bélanger et Dominic Thibault, jusqu'en 2011, année où Csound fût remplacé par pyo [1], un module python, créé par Olivier Bélanger, dédié au traitement de signal audio. L'utilisation d'un module python comme moteur audio, plutôt qu'un langage de programmation externe tel que Csound, offre certains avantages au niveau du design du logiciel. D'une part, l'intégration du moteur audio à l'interface de contrôle est grandement simplifiée en ce que les deux existent dans le même environnement de programmation. Pour communiquer avec Csound, les versions antérieures de Cecilia devaient avoir recours soit à une traduction pour python de l'API de Csound, soit au protocole de communication OSC [7]. Ces deux méthodes imposaient plusieurs contraintes en ce qui concerne le développement de nouveaux paradigmes de contrôle du processus audio. Avec un module audio dédié au langage python, le transfert des données de l'interface graphique vers les paramètres du processus sonore s'effectue sans intermédiaire, de façon plus directe que lorsque deux logiciels indépendants tentent de communiquer entre eux. Enfin, un langage de programmation orienté-objet tel que python offre beaucoup plus de flexibilité concernant l'écriture d'algorithmes audio et de contrôle, la gestion de la polyphonie ainsi que la manipulation en temps-réel des paramètres du son. Ce changement majeur dans l'architecture du logiciel a donné naissance à Cecilia 5, la version actuelle du logiciel.

Dans les versions précédentes, Cecilia était composé de deux éléments d'interface distincts. Une première fenêtre présentait un éditeur de texte, optimisé pour l'écriture de programmes Csound, permettant de développer des modules de traitement à même le logiciel. Ensuite, une seconde fenêtre contenant l'interface graphique de jeu, c'est-à-dire le grapheur, les potentiomètres et plusieurs autres

1 . *Musical Instrument Digital Interface*
2 . *Open Sound Control*
3 . *Application Programming Interface*

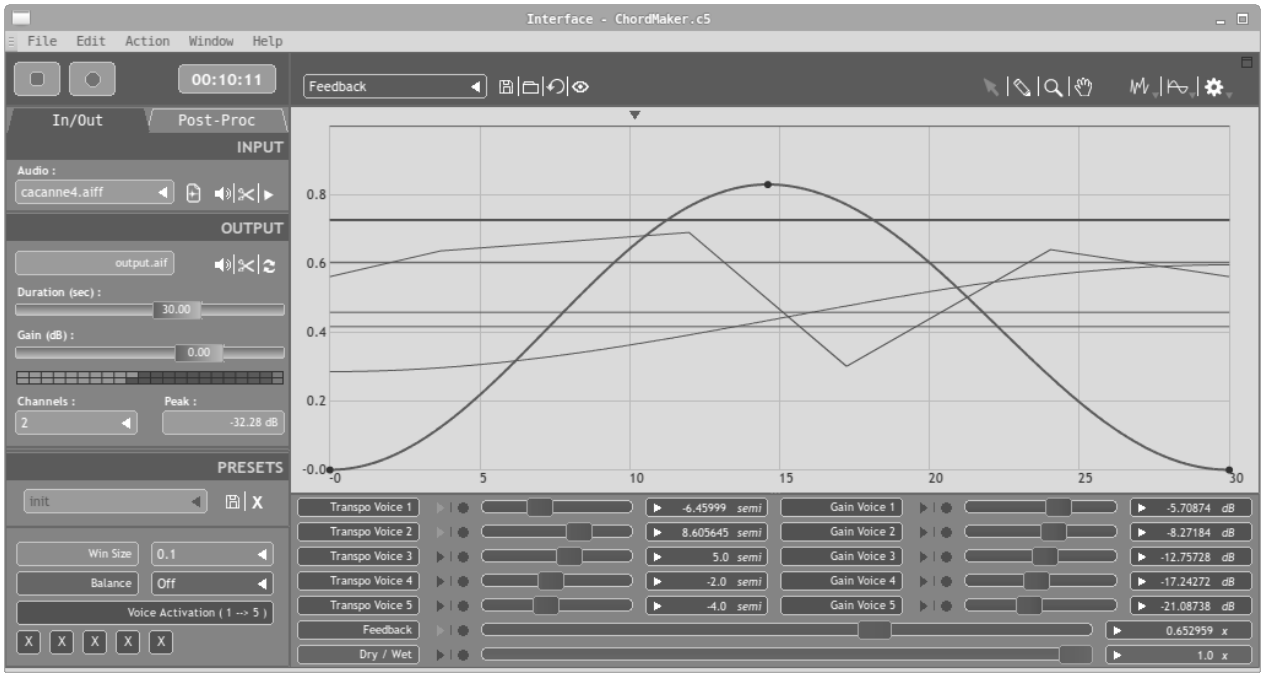

Figure 1. Fenêtre principale de l'interface graphique de Cecilia 5. On y retrouve différentes sections. En haut à gauche : le panneau de transport. Au centre à gauche : le panneau de contrôle avec les commandes d'entrée, de sortie et de sauvegarde des pré-réglages. En bas à gauche : la section des menus et boutons. Au centre de l'interface : le grapheur où sont éditées les automations de paramètres. Sous le grapheur : la section dédiée aux potentiomètres.

outils de contrôle, permettait de manipuler le processus audio en temps réel. Les modules de traitement audio de la version 5 étant écrit avec le langage python, il a été décidé d'alléger la structure du logiciel en retirant l'éditeur de texte. Il existe déjà une grande quantité d'éditeurs de texte gratuits et multi-plateformes, où le langage python est automatiquement reconnu, pouvant servir à l'écriture des modules. Olivier Bélanger est maintenant le développeur principal de Cecilia 5, alors que Jean Piché demeure le père spirituel du projet. Jean-Michel Dumas a grandement contribué à la traduction des modules Csound (de la version 4) en modules pyo pour la version 5.

2. UTILISATION DU LOGICIEL

L'objectif de base de l'application consiste à fournir un environnement simple et efficace afin d'explorer les effets audio-numériques sur le son. Une bibliothèque de modules, couvrant un large éventail de traitements, est fournie avec le logiciel. L'utilisateur, après avoir chargé un son dans le module choisi, peut manipuler et automatiser les différents paramètres du processus à l'aide des éléments d'interface graphique. Différentes fonctions d'enregistrement permettent de sauvegarder le résultat sonore sur le disque dur.

Dans cette section seront décrits les principaux éléments de l'interface graphique (Figure 1) et les possibilités de création qu'ils offrent à l'utilisateur. Nous y retrouverons les différents types de sources sonores à traiter, les modes d'enregistrement du rendu sur le disque dur, les techniques proposées pour moduler les paramètres au cours du temps et quelques utilitaires favorisant une expérience créative avec Cecilia.

2.1. Source sonore

La porte d'entrée du logiciel est la source sonore à modifier. Traditionnellement, Cecilia était conçu pour traiter un fichier son lu en boucle par l'application. Au fil des versions, la lecture en boucle du son a été grandement bonifiée, au point de devenir un élément important en tant que traitement audio dans le module. Divers modes de jeu, permettant notamment l'accès aux signaux provenant de la carte de son, ont aussi été ajoutés. Ces fonctionnalités, accessibles en cliquant sur l'icône à la droite du menu, font de Cecilia un environnement de traitement en temps réel simple d'approche et versatile. La configuration du mode de jeu et le choix de la source sonore s'effectue à l'aide de la section « Input » du panneau de contrôle.

Figure 2. Panneau de configuration de la source sonore.

2.1.1. Lecteur de son en boucle

Lorsqu'un son est chargé en mémoire dans le lecteur, la fenêtre de contrôle de la boucle peut être affichée en cliquant sur le petit triangle horizontal, dernier icône de la boîte d'outils à droite du menu (Figure 2).

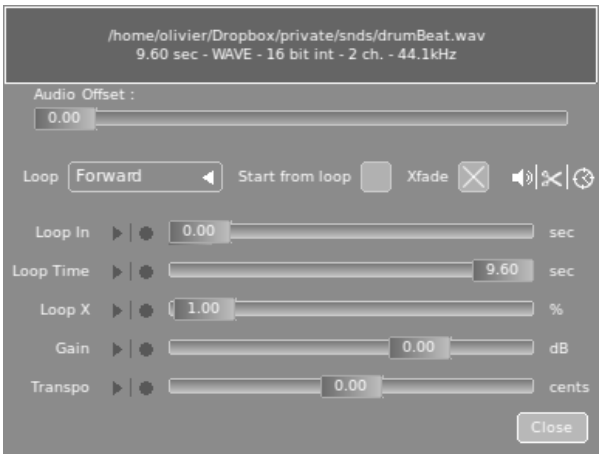

Figure 3. Fenêtre de contrôle du lecteur en boucle.

Cette fenêtre permet de spécifier les paramètres de la lecture, c'est-à-dire le point de départ de la boucle, sa durée, le temps de fondu-enchaîné aux points de bouclage, le volume du signal ainsi qu'un facteur de transposition en demi-tons. Tous ces paramètres peuvent être asservis à des contrôleurs externes, via le protocole MIDI ou le protocole OSC, et enregistrés dans le grapheur en tant qu'automations.

2.1.2. mode 1 : Fichier son

Le mode de jeu principal du logiciel (icône de fichier sonore) consiste à charger un fichier son, à l'aide du menu de la section « Input », dans la mémoire du lecteur. La lecture de ce son servira de signal source au traitement mis en place par le module choisi. Pour ouvrir un fichier son, il suffit de cliquer sur le menu, de naviguer dans la hiérarchie de dossiers à l'aide du dialogue présenté, puis de sélectionner le fichier désiré. En plus du fichier sélectionné, tous les fichiers sons contenus dans le même dossier viendront peupler le menu, offrant ainsi un accès rapide à une bibliothèque de sources sonores. On affiche le contenu en cliquant sur la petite flèche à droite du menu.

2.1.3. mode 2 : Entrée micro

Le deuxième mode de jeu (icône de micro) permet d'utiliser le signal en provenance des entrées de la carte de son comme source sonore à modifier. Configuré ainsi, Cecilia devient un environnement de traitement en temps réel à la fois simple d'utilisation et possédant des contrôles sophistiqués.

2.1.4. mode 3 : Entrée micro et lecteur en boucle

Le troisième mode de jeu (icône de micro accompagné du chiffre 1) permet à l'utilisateur d'enregistrer le signal d'entrée dans la mémoire du lecteur en boucle. La durée de la mémoire est spécifiée dans la fenêtre de contrôle de la boucle. Ce mode est utile pour manipuler un signal réel, qui n'a pas été préalablement enregistré sur le disque dur, tout en bénéficiant des fonctionnalités du lecteur en boucle.

2.1.5. mode 4 : Entrée micro continue et lecteur en boucle

Le dernier mode (icône de micro entouré de flèches circulaires) est une variante du mode précédent, où la mémoire du lecteur en boucle est constamment renouvelée par le signal réel en entrée de la carte de son. Pour éviter les artefacts causés par le bris de la forme d'onde au croisement des pointeurs de lecture et d'écriture, deux espaces-mémoire sont utilisés. La lecture et l'écriture alternent, hors phase, entre les deux mémoires. Le lecteur en boucle manipule donc un matériau en constante évolution.

2.2. Sortie audio

La configuration du signal de sortie s'effectue via la section « Output » du panneau de contrôle (Figure 4). On y retrouve, entre autres, le nom du fichier son à enregistrer, la durée totale de la performance (directement reliée aux automations du grapheur), un contrôle de volume global ainsi qu'un menu permettant de spécifier le nombre de canaux audio de sortie. Cette section est toujours présente, peu importe le module choisi pour traiter le signal. Les caractéristiques générales du signal, telles que la fréquence d'échantillonnage, la quantification, le choix des interfaces audio et MIDI ou le format audio désiré peuvent être modifiées par le biais de la fenêtre de préférences du logiciel.

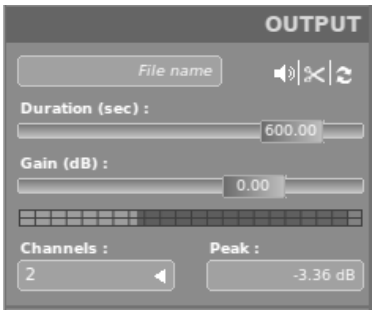

Figure 4. Panneau de configuration de la sortie audio.

Divers modes d'écoute et de sauvegarde sont disponibles pour rendre le travail dans l'environnement efficace.

2.2.1. Panneau de transport

Le panneau de transport (Figure 5) est situé dans le coin supérieur gauche de l'interface. Il permet d'activer la per-

formance, à l'aide du bouton *play*, pour la durée totale spécifiée dans la section « Ouptut ». Le bouton d'enregistrement, en plus d'activer la performance, enregistre en temps réel le signal de sortie sur le disque dur. Lorsque la performance est lancée en temps réel, le curseur situé au-dessus du grapheur entreprendra sa course et toutes les manipulations effectuées dans l'interface (menus, boutons, potentiomètres, lignes de graphe) auront des répercutions immédiates sur le signal de sortie.

Figure 5. Panneau de transport.

2.2.2. *Écriture sur le disque*

Pour les performances de longues durées, où l'évolution des paramètres est entièrement configurée à même le grapheur, il y a possibilité de sauvegarder le résultat sonore, en temps différé, dans un fichier sur le disque dur. Cecilia calculera alors le signal le plus vite possible, en fonction de la puissance du processeur. Cette fonction est accessible sous le menu *Action* → *Bounce to Disk*.

2.2.3. *Exportation par lot*

Plusieurs fichiers sonores peuvent être créés automatiquement avec les fonctions d'exportation par lot. Deux méthodes d'exportation sont disponibles. La première, accessible sous le menu *Action* → *Batch Processing on Preset Sequence*, permet d'appliquer tous les pré-réglages enregistrés (voir section 2.4.2), un par fichier généré, sur le fichier sonore courant. Cette fonction est très utile pour créer une séquence de sonorités ayant une source commune. La seconde méthode, activée via *Action* → *Batch Processing on Sound Folder*, permet d'appliquer le processus courant sur tous les sons chargés dans le menu, avec option d'ajuster automatiquement la durée des sons générés sur la durée des sons originaux. Cette fonction est particulièrement utile lorsque plusieurs sons doivent être traités de façon similaire.

2.3. Modulation des paramètres au cours du temps

Une des grandes forces de Cecilia réside dans les différents paradigmes offerts pour la génération de trajectoires de paramètres. Cette section exposera les principaux outils permettant de créer des traitements audio originaux et dynamiques.

2.3.1. *Le grapheur*

Le grapheur (Figure 6) est l'élément central de l'application. Tous les paramètres continus (potentiomètres inclus) y sont représentés sous la forme d'une ligne de couleur différente. Lorsque la performance est lancée, chacun des paramètres dont l'automation est activée se verra soumis à la trajectoire qui lui est associée dans le grapheur.

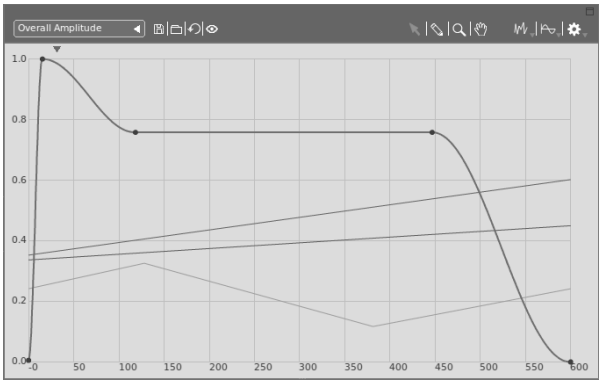

Figure 6. Gestion de l'évolution des paramètres au cours du temps.

La barre d'outils situé au-dessus du grapheur est divisée en trois sections :

1. La section de gauche (Figure 7) offre les outils pour la gestion des trajectoires dans le grapheur. On y retrouve le menu de sélection, la sauvegarde et l'ouverture de fichier, un bouton de ré-initialisation ainsi qu'un commutateur de visibilité.

Figure 7. Gestion des trajectoires du grapheur.

2. La section du centre (Figure 8) permet la sélection des outils de manipulation graphique. Le pointeur permet de déplacer, sélectionner et créer des points sur une trajectoire. Le crayon permet de dessiner des courbes en mode « main libre ». La loupe permet d'agrandir une section du grapheur afin d'effectuer des manipulations détaillées sur une portion de trajectoire. Enfin, en mode agrandi, la main permet de déplacer la région visible du grapheur.

Figure 8. Outils de manipulation du grapheur.

3. La section de droite (Figure 9) donne accès à divers algorithmes de génération ou de manipulation de trajectoire. On y retrouve des algorithmes de génération aléatoire (voir section 2.3.2), des générateurs de formes d'onde diverses ainsi qu'une série d'algorithmes de manipulation de trajectoire. Ces algorithmes permettent notamment de compresser (ou de dilater) un groupe de points, d'interpoler entre deux points ou d'ajouter des déviations aléatoires.

Figure 9. Générateurs et modificateurs de trajectoire.

2.3.2. Les générateurs algorithmiques

Les générateurs algorithmiques, dont les contrôles sont illustrés sur la figure 10, offre à l'utilisateur une méthode simple et puissante pour créer des trajectoires aléatoires aux polarités influencées par l'algorithme choisi. Un choix de courbes, telles que uniforme, gaussienne, weibull et beta, ainsi que des algorithmes de génération musicale, comme la marche aléatoire, les répétitions et les segments bouclés, sont disponibles via le premier menu de la fenêtre. Deux types d'interpolation sont offerts dans le second menu. Il y a l'interpolation linéaire, où chacun des points est relié au suivant par une ligne droite, et le *sample and hold*, où la valeur d'un point est tenue jusqu'au point suivant. Viennent ensuite une série de potentiomètres permettant de spécifier la quantité de points à générer, les balises minimum et maximum à l'intérieur desquelles la génération sera limitée ainsi que les paramètres de contrôle liés à l'algorithme sélectionné. Les trajectoires générées constituent un excellent point de départ pour une paramétrisation dynamique d'un processus musical.

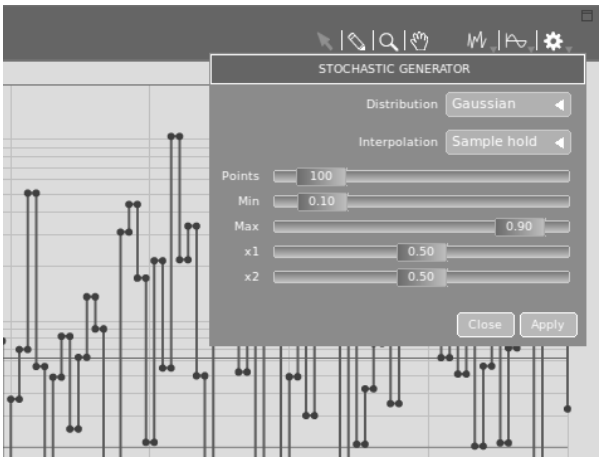

Figure 10. Génération algorithmique de trajectoires.

2.3.3. Automation de paramètres

Tous les paramètres continus, qu'ils appartiennent au module lui-même, au lecteur de son en boucle ou aux effets post-traitement (voir section 2.4.1), peuvent être asservis à un contrôleur externe via le protocole MIDI ou le protocole OSC. La figure 11 illustre l'assignation de canaux OSC aux fréquences de coupure de deux filtres ainsi que l'assignation d'un contrôleur MIDI sur le paramètre de mixage des filtres. Les connexions sont indiquées sur le fond du potentiomètre et sauvegardées à même le module lorsque celui-ci est enregistré. On double-clique sur l'étiquette du potentiomètre pour ouvrir la fenêtre d'assignation de canaux OSC ou on clique avec le bouton droit pour lancer la détection automatisée de contrôleur MIDI (fonction *midi learn*).

Figure 11. Assignation de contrôleurs externes aux potentiomètres.

2.4. Les utilitaires

Parmi les fonctionnalités présentes dans le logiciel, il y en a deux qui permettent à l'utilisateur de personnaliser les processus appliqués au signal source. D'abord, l'onglet « Post-Processing » du panneau de contrôle expose une interface où des effets supplémentaires peuvent être appliqués sur le signal audio produit par le module. Ensuite, un système de pré-réglages permet de sauvegarder indépendamment plusieurs états du module et de les rappeler à volonté.

2.4.1. Post-traitement

Le panneau de post-traitement met à la disposition de l'utilisateur un certain nombre d'effets, aux contrôles simplifiés, à la manière des *plugins* dans un séquenceur. Ces effets, entièrement automatisables dans le grapheur, sont appliqués en série sur le signal produit par le module courant. Ce système permet notamment de personnaliser un module en lui ajoutant une ou plusieurs couches de traitements supplémentaires. On y retrouve un éventail d'effets classiques, tels que la réverbération, le filtrage, la distorsion, l'harmonisation, la compression, etc.

Figure 12. Panneau des traitements post-module.

2.4.2. Pré-réglages

La section « Presets » (Figure 13) permet de sauvegarder l'état du module en plusieurs moments différents afin de pouvoir naviguer rapidement d'un état à l'autre. Ces multiples pré-réglages sont sauvegardés à même le module (c'est-à-dire le fichier portant l'extension « c5 », où est définit la chaîne de traitement de signal ainsi que l'interface graphique correspondante) lorsque ce dernier est enregistré. On navigue d'un pré-réglage à l'autre en les sélectionnant dans le menu déroulant. Ce système fait en sorte qu'un seul module Cecilia peut devenir un environnement de travail créatif autonome et complet. Un processus sonore peut alors être développé en plusieurs étapes, chacune de ces étapes ayant un rôle spécifique à jouer dans une construction musicale donnée. Tout ce processus est sauvegardé dans un seul fichier texte, entièrement portable d'un système d'exploitation à un autre.

Figure 13. Panneau de gestion des pré-réglages.

3. ÉCRITURE DE MODULES CECILIA

Depuis la version 5 de Cecilia, un module n'est rien d'autre qu'un simple script python, contenant une classe où est décrite la chaîne de traitement de signal et une liste d'éléments d'interface graphique à afficher. En plus de la librairie de modules fournis avec l'application, des modules originaux peuvent être développés avec un simple éditeur de texte et ouverts directement par Cecilia. L'API de Cecilia est accessible à même le logiciel, via le menu *Help → Show API Documentation*. Il contient principalement deux sections. La première concerne la classe audio où est implémenté le processus sonore. La seconde section décrit les différents outils graphiques disponibles pour la construction de l'interface.

3.1. La classe de traitement audio

Un module Cecilia doit absolument contenir une classe nommée « Module », dans laquelle sera développée la chaîne de traitements sonores. Cette classe, pour fonctionner correctement dans l'environnement Cecilia, doit hériter de la classe « BaseModule », définie à même le code source de Cecilia. C'est la classe « BaseModule » qui se charge de créer, à l'interne, les liens entre l'interface graphique, les contrôleurs externes et le processus audio lui-même.

3.1.1. Initialisation

La classe audio d'un module doit impérativement être déclarée comme ceci :

```
class Module(BaseModule):
    def __init__(self):
        BaseModule.__init__(self)
        # traitement sur le signal...
```

Comme le fichier sera exécuté à l'intérieur de l'environnement Cecilia, il n'est pas nécessaire d'importer *pyo* ou la classe « BaseModule », à laquelle on fait référence en tant que classe parente. Ces composantes étant déjà importées par Cecilia, elles seront disponibles à l'exécution du fichier.

3.1.2. Sortie audio du module

Afin d'acheminer le signal audio du processus à l'application, « self.out » doit absolument être le nom de variable de l'objet à la toute fin de la chaîne de traitement. Cecilia récupère cette variable pour faire suivre le signal audio vers la section post-traitement et ultimement, à la sortie audio. Un example typique de déclaration de la variable « self.out » ressemblerait à ceci :

```
self.out = Mix(self.boo, self.nchnls, self.env)
```

Le signal modifié, « self.boo », est remixé en fonction du nombre de canaux spécifié par l'interface de Cecilia (« self.nchnls »), puis pondéré en amplitude par la ligne de graphe « self.env ».

3.1.3. Attributs et méthodes publiques de la classe « BaseModule »

Certaines variables publiques, définies à l'initialisation de la classe « BaseModule », contiennent des informations importantes sur la configuration courante de Cecilia et peuvent être utilisées à tout moment dans la gestion du processus audio. De même, la classe « BaseModule » définit un certain nombre de méthodes utilitaires qui peuvent être utilisées dans la composition de la chaîne de traitements du module.

Principaux attributs publics :

- **self.sr** : Fréquence d'échantillonnage de Cecilia.
- **self.nchnls** : Nombre de canaux audio en sortie.
- **self.totalTime** : Durée totale de la performance.

Principales méthodes publiques :

- **self.addFilein(name)** : Charge un son dans un espace mémoire.
- **self.addSampler(name, pitch, amp)** : Génère un lecteur de son en boucle.
- **self.getSamplerDur(name)** : Retourne la durée, en secondes, du son chargé dans un lecteur.

Le lecteur est invité à consulter la documentation de Cecilia pour obtenir la liste complète des attributs et méthodes disponibles.

3.1.4. Documentation du module

Les informations pertinentes à une bonne compréhension du comportement d'un module peuvent être données dans le « __doc__ string » de la classe « Module ». L'utilisateur de Cecilia peut, à tout moment, consulter la documentation du module via la commande menu *Help* → *Show module info*.

3.2. Déclaration des éléments d'interface

Le fichier Cecilia (portant l'extension « c5 »), en plus de la classe audio, doit spécifier la liste des contrôles graphiques nécessaires au bon fonctionnement du module. Cette liste doit impérativement porter le nom « Interface » et être constituée d'appels de fonctions propres à l'environnement Cecilia. Voici un exemple où est déclaré un lecteur de son, une ligne de graphe, trois potentiomètres et un contrôle de la polyphonie :

```
Interface = [
    csampler(name="snd"),
    cgraph(name="env", label="Overall Amplitude",
           func=[(0,1),(1,1)], col="blue1"),
    cslider(name="drv", label="Drive", min=0, max=1,
            init=0.5, col="purple1"),
    cslider(name="lp", label="LP F", min=20, max=15000,
        rel="log", init=5000, unit="Hz", col="green1"),
    cslider(name="q", label="LP Q", min=0.5, max=5,
            init=1, col="green2"),
    cpoly()
]
```

Ces quelques lignes de code suffisent à produire l'interface graphique suivante :

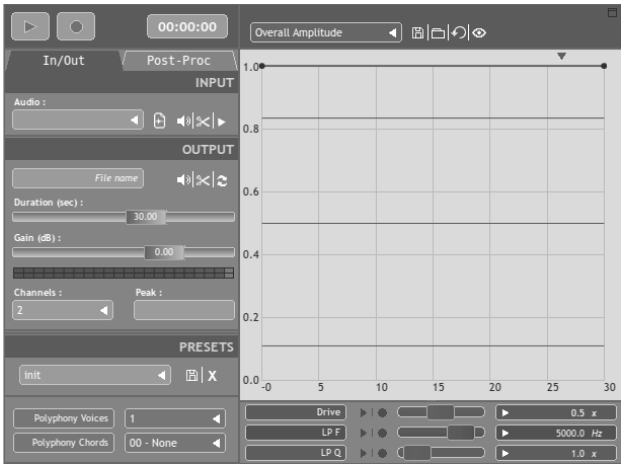

Figure 14. Interface graphique (les outils de manipulation du grapheur sont cachés sur cette figure).

La documentation de Cecilia fournit la liste complète des fonctions graphiques disponibles et de l'élément d'interface qu'elles permettent de créer.

3.3. Exemple

Le code suivant, basé sur l'interface graphique générée à la section précédente, illustre la création d'un module de distorsion du signal agrémenté d'un filtre passe-bas résonant.

```
class Module(BaseModule):
    """
    Module's documentation

    """
    def __init__(self):
        BaseModule.__init__(self)
        self.snd = self.addSampler("snd")
        self.pre = Sig(self.drv, mul=10, add=1)
        self.clp = Clip(self.snd*self.pre, -.99, .99)
        self.flt = Biquad(self.clp, self.lp, self.q)
        self.boo = self.flt * 0.2
        self.out = Mix(self.boo, self.nchnls, self.env)


Interface = [
    csampler(name="snd"),
    cgraph(name="env", label="Overall Amplitude",
           func=[(0,1),(1,1)], col="blue1"),
    cslider(name="drv", label="Drive", min=0, max=1,
            init=0.5, col="purple1"),
    cslider(name="lp", label="LP F", min=20, max=15000,
        rel="log", init=5000, unit="Hz", col="green1"),
    cslider(name="q", label="LP Q", min=0.5, max=5,
            init=1, col="green2"),
    cpoly()
]
```

4. CONCLUSION

Grâce à son large éventail de modules de traitement de signal et à son interface graphique simple et versatile, Cecilia constitue un environnement de travail idéal pour l'exploration et la production sonore. Les nouveaux modes de jeu, permettant d'injecter dans le module le signal en provenance de la carte de son, alliés au contrôle des paramètres via des interfaces externes (MIDI ou OSC), font de Cecilia un environnement de traitement en temps réel aux multiples possibilités. La simplicité de l'API de Cecilia en fait un outil pédagogique motivant pour l'apprentissage de la programmation musicale et devrait favoriser, dans un futur rapproché, la création de modules d'effets audio-numérique originaux.

5. REFERENCES

[1] Bélanger, O., « La programmation audio avec Python », *GNU Linux Magazine France*, éditions diamond, vol. 157, France, 2013.

[2] Cecilia5, « Projet Cecilia5 sur googlecode », *http ://code.google.com/p/cecilia5/*, (2011-2014).

[3] Dunn, R., Rappin, N., « wxPython in action », Manning publications, Greenwich, CT, 2006.

[4] Piché, J., Burton, A., « Cecilia : A Production Interface to Csound », *Computer Music Journal*, MIT Press, 1998, Vol 22, numéro 2, p. 52–55.

[5] Python S. F., « Python programming language – official website », *http ://python.org/*, (1990-2013).

[6] Vercoe, B., Ellis, D., « Real-time csound : Software synthesis with sensing and control », *International computer music conference*, 1990, p. 209–211, Glasgow.

[7] Wright, M., et al., « OpenSound Control : State of the Art 2003 », *Conference on New Interfaces for Musical Expression*, 2003, Montréal.