# COMPOSITION ASSISTÉE PAR ORDINATEUR : LE CAS DE EJS

Joakim Sandgren

▶ **To cite this version:**

Joakim Sandgren. COMPOSITION ASSISTÉE PAR ORDINATEUR : LE CAS DE EJS. Journées d'Informatique Musicale, Mar 2008, Albi, France. hal-03052888

**HAL Id: hal-03052888**
**https://hal.science/hal-03052888**

Submitted on 10 Dec 2020

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# COMPOSITION ASSISTÉE PAR ORDINATEUR : LE CAS DE *EJS*

Joakim Sandgren
CICM
Université Paris VIII
MSH Paris Nord
info@joakimsandgren.com

## 1. RÉSUMÉ

Dans ce texte je présente, d'une façon encore très incomplète, l'environnement de composition *ejs*. D'abord je mets en contexte cet article. Puis j'aborde les questions qui me mènent à ne pas travailler avec les logiciels existants. Ensuite je présente *ejs*, sa structure informatique, et je montre une vue d'ensemble du flux de travail. Finalement j'étudie les fonctions principales et le travail dans le script.

## 2. INTRODUCTION

Ce travail se situe dans un cadre d'études doctorales qui visent le développement d'un environnement informatique de composition musicale en lisp. Cet environnement a pour premier objectif de servir une pensée compositionnelle personnelle.

Ce travail de recherche, de développement et de création musicale a lieu au sein de l'école doctorale "esthétique, sciences et technologies des arts" de l'université de Paris VIII. Il se développe dans le champ universitaire des sciences humaines, là où, en art, et spécialement en création musicale contemporaine, les artistes chercheurs expérimentent et s'approprient les nouvelles technologies en interagissant avec leur développement, afin de les mettre au service de leur pratique, et afin d'approfondir plus largement une réflexion sur leur "manière de faire".

Ce texte représente le début de mon travail de compositeur-chercheur, qui intègre d'une part des développements logiciels et d'autre part des compositions musicales créées à un niveau international[1].

## 3. *EJS*

### 3.1 Pourquoi *ejs*.

Il y a plusieurs logiciels avec lesquels on peut composer "assisté" par un ordinateur, entre autres OpenMusic [1], PWGL [8], Elody [12] et Common Music [19]. Je propose de démontrer pourquoi j'utilise mon propre environnement.

Depuis la famille Music de Max Mathews dans les années 60 à 70, des logiciels musicaux existent. Ces logiciels étaient utilisés par un petit nombre de compositeurs qui avaient la capacité de concevoir leur musique en écrivant du code de programmation, ou au moins des scripts.

Avec les développements dans le domaine informatique, et surtout le développement du coté graphique dans les années 80 à 90, il est devenu possible de représenter des fonctions avec une interface graphique. Cela a permis de s'extraire du code de programmation, pour exprimer les mêmes idées avec des symboles graphiques dans un langage visuel, ce qui a eu pour effet de rendre accessible toute la puissance informatique aux compositeurs "normaux" qui n'avaient pas l'habitude d'écrire du code de programmation. C'est cela le projet, l'idée de PatchWork [9], de PWGL, de Elody et de OpenMusic : de supprimer la présence du code en le remplaçant par des symboles graphiques, maniables, compréhensibles et ainsi accessibles à tous les compositeurs. Si on regarde les présentations de quelques-uns de ces logiciels, ils se manifestent fermement dans la tradition de la programmation visuelle ;

- OpenMusic: *OpenMusic (OM) est un langage de programmation entièrement visuel basé sur CommonLisp/ CLOS.*[2]
- PWGL: *PWGL is a free cross-platform visual language based on Common Lisp, CLOS and OpenGL.*[3]
- Elody: *Elody est un environnement de composition musicale basé sur un langage fonctionnel visuel...*[4]

---

[1] *pour un(e) flûtiste à bec et neuf musiciens*, 26/4/2005, Malmö Suède, Ars Nova ; *pour trois musiciens*, 15/5-2005, Strasbourg, In Extremis ; *pour un(e) pianiste*, 25/5-2005, Saint-Denis, Imma Santacreu ; *pour un(e) pianiste*, 26/5-2005, Théatre Philippe Gérard, Saint Denis, Imma Santacreu ; *pour un (e) pianiste*, 27/5-2005, Institut Cervantes, Paris, Imma Santacreu ; *pour un(e) pianiste*, 4/9-2005, Stockholm Suède, Eva Sidén ; *pour un(e) pianiste*, 4/10-2005, Stockholm, Eva Sidén, *pour un(e) pianiste*, 4/2-2006, Paris, Eva Sidén *; pour sept musiciens*, 18/6-2006, Villemomble, l'Ensemble Archaeus ; *instrument contondant*, 25/2-2007, Göteborg, l'Ensemble Gageego! *; lignes granulées pour un(e) pianiste*, 15/4-2007, Musée Modern, Stockholm, Eva Sidén ; *encore pour un(e) flûtiste à bec*, 22/4-2007, Vienne, Kerstin Frödin ; *pour un(e) flûtiste à bec et neuf musiciens*, 1/9-2007, Norrköping, Suède, l'Ensemble Ma.

[2] http://freesoftware.ircam.fr/rubrique.php3?id_rubrique=6

[3] http://www2.siba.fi/PWGL/

[4] http://www.grame.fr/Elody/

Quant à moi, j'ai commencé à composer "assisté par ordinateur" en 1997 avec PatchWork, avec Magnus Lindberg qui enseignait à Stockholm à l'époque. J'avais vu que la façon dont je travaillais s'adaptait parfaitement à la façon dont on travaille dans PatchWork, et j'ai commencé à composer en faisant des patchs. Pourtant, je me suis très rapidement mis à exprimer les mêmes patchs en code Lisp. Très vite, j'étais en train de programmer ma propre bibliothèque PatchWork, *jsm-gesture* [12], avec laquelle j'ai composé ma première pièce "CAO" [13]. En fait, j'ai appris le langage Lisp par les patchs de PatchWork. À l'époque, il y avait, en effet, de la vitesse à gagner en utilisant du code directement, au lieu des patchs, mais ce n'était pas cela ma motivation. Je *pensais* mieux le problème musical en question en écrivant du code qu'en faisant des patchs. Je me perdais dans mes patchs, mais une fois les patchs traduits en code je les "voyais" mieux. C'était donc un gain pour moi d'écrire du code. Pour la pièce suivante, je me suis mis directement sur la plate-forme MCL [4] et j'ai programmé mon propre projet, qui aboutissait à une pièce pour ensemble [14]. Dans mon cas, c'est donc une question de prédisposition personnelle qui m'a rendu le code plus compréhensible que le patch.

Quant à l'environnement Common Music, qui a comme *ejs* une interface textuelle, il est plus ciblé sur le son :

- *Common Music (CM) is an object-oriented music composition environment. It produces sound by transforming a high-level representation of musical structure into a variety of control protocols for sound synthesis and display*[5].

Or, à l'époque je travaillais uniquement avec du MIDI.

Je considère tous ces logiciels et le cheminement qu'ils représentent très importants, mais je ne m'y suis tout simplement pas retrouvé et je suis revenu à l'ancienne interface. Aujourd'hui, mon propre projet logiciel est devenu assez vaste et ne pourrait de toute façon pas se faire avec des patchs à cause de son ampleur. Il est devenu un environnement dit "expert" avec lequel je pense ma musique. Des enregistrements et des partitions de pièces écrites avec *ejs* sont disponibles sur :
http://joakimsandgren.com puis -> *mp3s*
http://joakimsandgren.com puis -> *scores* (sous format de pdf).

### 3.2 Présentation de *ejs*

L'environnement *ejs* est un environnement informatique expert de composition réalisé en Common Lisp[6] et CLOS[7] sur la plate-forme MCL. Il est fondé sur l'idée du façonnement de toute une composition de musique instrumentale écrite, avec la possibilité de sa simulation immédiate et comprenant l'engendrement d'une partition instantanée.

Avec *ejs* on travaille obligatoirement la structuration de toute une composition dans son intégralité.

Le travail se fait dans un environnement textuel, avec des appels de fonction. On fait une centaine d'appels pour réaliser le déroulement d'une voix dans une pièce. J'appelle cela le *script*.

Lors du travail de composition, on se déplace constamment entre différentes sections du *script* qui sont réparties sur plusieurs documents différents, pour s'orienter plus facilement dans cette quantité considérable de texte.

Comme on ne peut que travailler une pièce entière, il faut toujours commencer par une constellation de paramètres. Comme il n'y a pas de paramètres véritablement "par défaut" on commence "quelque part", peut-être avec des idées qu'on a pour la composition en question. Je prends souvent le *script* - les documents - d'un *instrument* d'une composition déjà écrite comme point de départ.

### 3.3 La structure informatique de *ejs*

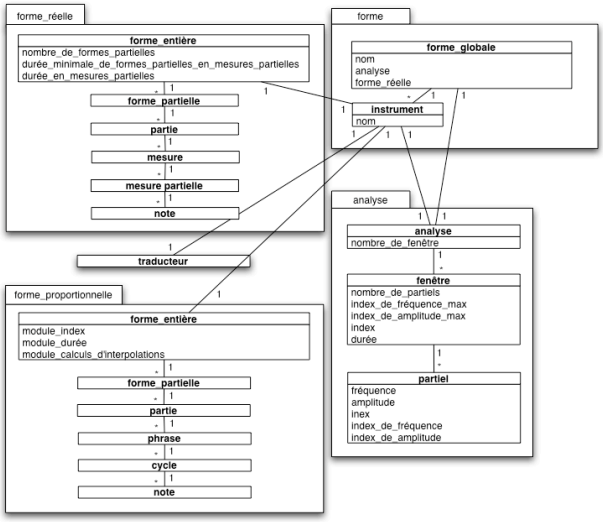


**Fig. 1** La structure informatique d'une partie de *ejs* (tous les champs d'objet ne sont pas affichés).

*ejs* est composé de plusieurs modules informatiques dont je ne montre ici que ceux qui nous concernent : *analyse*, *forme*, *forme_réelle* et *forme_proportionnelle*.

Je vais dans ce texte me référer à une instance de la classe 'forme_globale' comme *la forme globale,* à une instance de la classe 'analyse' comme *l'analyse,* à une instance de la classe 'forme_entière' du module *forme_proportionnelle* comme *la forme proportionnelle,* à une instance de la classe 'forme_entière' dans le module *forme_réelle* comme *la forme*

[5] http://commonmusic.sourceforge.net/doc/cm.html
[6] Langage fonctionnel, dynamiquement typé.
[7] CLOS : Common Lisp Object System, système pour une programmation orientée objet.

*réelle,* et à une instance de la classe 'instrument' comme *l'instrument*.

La composition est regroupée dans un objet unique, *la forme globale*. Toutes les données de la pièce, formelles et autres, sont fournies par *l'analyse*. Même si le travail de composition se passe principalement en travaillant *la forme proportionnelle,* il faut commencer par définir les données "réelles" de la pièce. C'est-à-dire, une partition avec la durée de la pièce, avec les tempos, les changements de mesures, etc. A partir de *l'analyse* je crée donc une partition "vide", sans notes et pauses. Cet objet est ensuite copié dans chaque *instrument*. Puis, pour chaque *instrument* je calcule le nombre maximal de notes ou pauses pour traduire la durée globale en unités proportionnelles utilisées par *la forme proportionnelle*. Une fois cela fait, je façonne avec le script *la forme proportionnelle* qui sera ensuite traduite et remise dans la *forme réelle*, la partition.

La hiérarchie d'objets que l'on peut voir dans *la forme proportionnelle* dans la Fig. 1 est appelée *la forme multi-couche*. Par cela j'entends la trame du réseau formel de *la forme proportionnelle*. Il s'agit d'un emboîtement successif sur cinq niveaux (Fig.2). Ces niveaux représentent cinq échelles temporelles allant du domaine de la forme globale au domaine rythmique. Ensuite un paramètre, comme la hauteur ou la dynamique, peut changer de niveau pour ainsi créer un chemin composé sur plusieurs échelles temporelles.

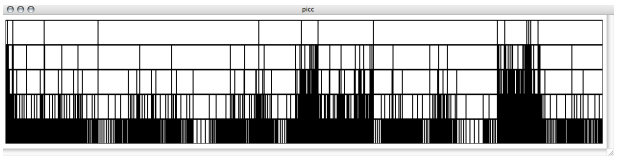

**Fig. 2** *Une forme multi-couche.*

## 4. UNE VUE D'ENSEMBLE DU FLUX DE TRAVAIL DANS *EJS*

Je propose maintenant de montrer comment je construis un *instrument* d'un bout à l'autre pour ainsi donner une vue d'ensemble du flux de travail dans *ejs*.

Tout commence à partir d'un son enregistré en mono.

En utilisant le logiciel AudioSculpt [6] je fais une "Chord Sequence Analysis" qui fournit une série de spectres avec leurs durées.

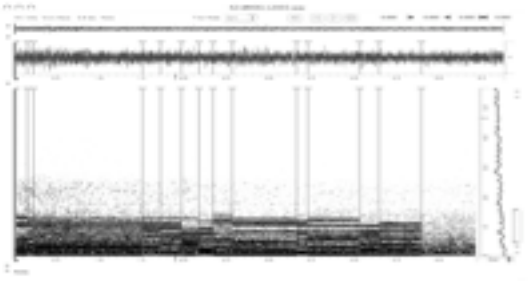

**Fig. 3** Une "Chord Sequence Analysis" dans Audiosculpt.

Ces spectres sont exportés vers un fichier texte. Ce fichier contient pour chaque partiel de chaque spectre le temps de début, la fréquence de début, l'amplitude de début, puis le temps de fin, la fréquence de fin et l'amplitude de fin (Fig. 4).

Maintenant commence le travail dans *ejs*. Je lis ce fichier texte et je le transforme en une *analyse*.

*L'analyse,* lors de sa création, indexe toutes les données du fichier texte : les durées, les fréquences et les amplitudes.

```
( PARTIALS   1045
( POINTS     2
   0.155          37.331         -17.116
   0.593          37.331         -17.116
)
( POINTS     2
   0.155          71.628         -25.743
   0.593          71.628         -25.743
)
```

**Fig. 4** Le contenu du fichier texte avec les données de la "Chord Sequence Analysis".

À partir de *l'analyse* je façonne maintenant *la forme proportionnelle* avec des appels de fonction (Fig. 5).

J'obtiens des valeurs depuis *l'analyse* et avec ces valeurs je crée des objets. Puis j'obtiens encore des valeurs et je les mets dans ces objets pour créer de nouveaux objets et ainsi de suite. Je peux aussi reprendre des valeurs déjà écrites dans *la forme proportionnelle* pour les retravailler et les remettre ailleurs.

Les données viennent donc soit de *l'analyse* soit des valeurs déjà écrites dans *la forme proportionnelle*. Il faut également noter que, même quand je cherche des données "fraîches" dans *l'analyse*, je les obtiens *à partir* de *l'analyse* mais *en fonction* des structures déjà écrites dans *la forme proportionnelle*.

Cela est un processus qui peut durer longtemps parce que c'est ici, avec cette façon de placer, reprendre, retravailler et replacer les données que le véritable travail de composition a lieu.

```
[écrire_champ
 :champ (définir_portée
          :champ 'nombre_de_formes_inférieures
          :niveau "formes_entières"
          :nom_dinstrument "elfi1"
          :print nil)
 :valeurs 15]
[écrire_champ
 :champ (définir_portée
          :champ 'proportion
          :niveau "formes_partielles"
          :nom_dinstrument "elfi1")
 :valeurs (créer_valeurs
           :nom_dinstrument "elfi1"
           :index (index "elfi1" 0 '() '(1)) :print t
           :champ 'durée
           :retourner_également 'index
           :traitements_de_valeurs
           (travail :traiteur 'sm::contraste-à-la-même-somme
                    :args (arg 1.0))))
```

**Fig. 5** Exemple du script. Deux appels de fonction.

Pour faciliter le travail avec *la forme proportionnelle* j'utilise quelques fenêtres de visualisation. Ce sont de simples moyens de visualisation sans aucune possibilité d'interaction avec la structure visualisée.

J'ai deux types de fenêtres. Dans le premier je visualise soit uniquement la ligne d'un paramètre, comme les hauteurs, ou la dynamique, soit la ligne avec *la forme multi-couche* en dessous (Fig. 8).

Dans le deuxième type de fenêtre je visualise les hauteurs et la dynamique dans des cases qui mettent en évidence en même temps la hiérarchie *multi-couche*. Ici, je peux également cliquer une case pour ainsi inspecter l'objet en question dans l'inspecteur de MCL.

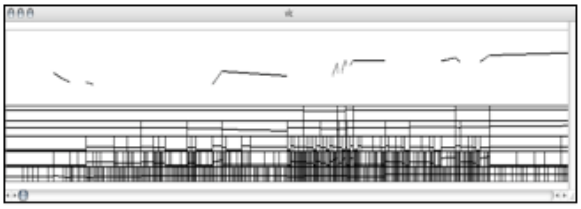

**Fig. 6** Fenêtre de visualisation du type 1 de *la forme proportionnelle*.

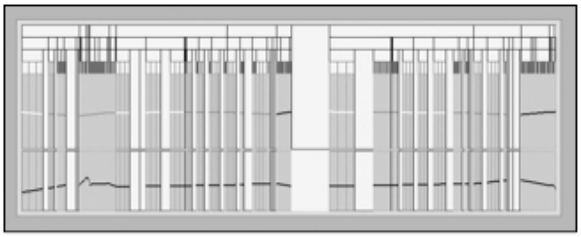

**Fig. 7** Fenêtre de visualisation du type 2 de la *forme proportionnelle*.

J'utilise des couleurs pour mettre en évidence les différents types musicaux comme : "silence", ou "son" ainsi comme des articulations différentes telles que "trémolo_lent" ou "spiccato", et je visualise de cette façon les caractéristiques musicales de la *forme multi-couche*.

Je peux également visualiser le résultat des fonctions principales sous forme de graphe. J'utilise le module *graph* fait par Richard Sutton [17].

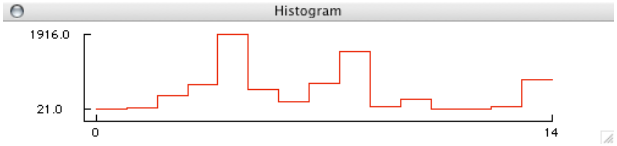


**Fig. 8** Graphe simple en forme d'histogramme pour visualiser des données.

J'ai également une fenêtre de visualisation de type deux pour *la forme réelle* où je peux voir les variations de tempo.

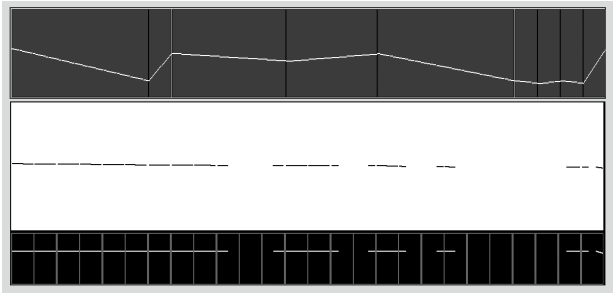

**Fig. 9** Fenêtre de visualisation du type 2 de la *forme réelle*. De haut en bas : le tempo ; les hauteurs ; la dynamique.

C'est à partir de *la forme réelle* que je peux simuler le résultat ou le regarder en forme de partition. C'est donc presque lors de chaque essai et de chaque changement du *script* que je traduis *la forme proportionnelle* en *une forme réelle,* pour ainsi en vérifier le résultat.

Ce n'est donc pas "une fois le travail de composition terminé" avec *la forme proportionnelle* que je la traduis mais plutôt tout le temps. Même si c'est effectivement avec *la forme proportionnelle* que je travaille et que c'est *la forme proportionnelle* qui contient la plupart de la composition et qui est évaluée la première et que *la forme réelle* est une forme vide sans *la forme proportionnelle,* les deux formes existent toutes les deux tout le temps.

Je peux exporter la pièce à partir de la forme réelle en plusieurs formats : en MIDI, en partition instantanée (cmn) et en xml pour importation dans *Finale* [10] ou *Sibelius* [18].

L'exportation en MIDI se fait avec le logiciel *OXSp* créé par Fredrik Hedelin [7].

Pour l'exportation de la partition instantanée j'utilise le module *Common Music Notation,* dit *cmn*, fait par Bill Schottstedt à Stanford University [16].

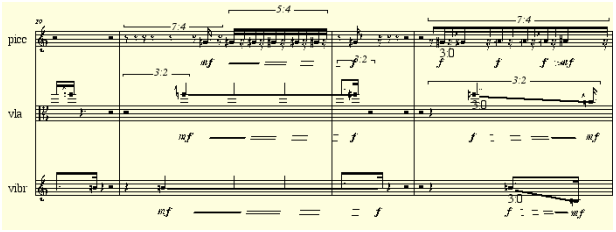


**Fig. 10** Partition instantanée engendrée utilisant le module *cmn*.

Pour l'exportation en xml j'utilise le nouveau format MusicXML inventé par Michel Good [3] et développé par lui à Recordare Inc. [4] MusicXML est considéré comme le format d'échange musical d'avenir, et une version *light* de leur plug-in Dolet [2] est inclus et dans Finale et dans Sibelius.

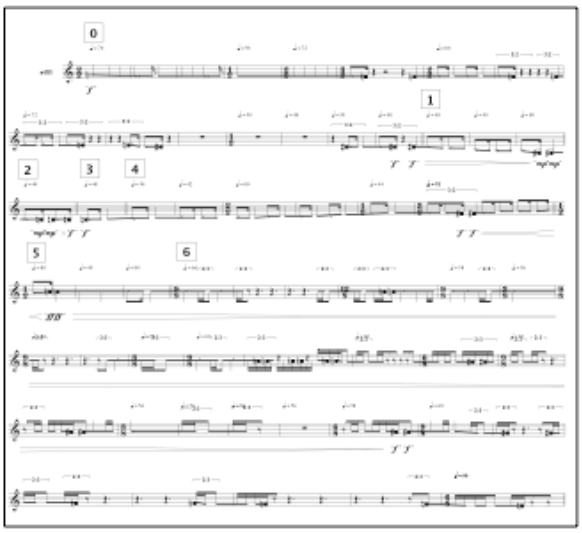

**Fig. 11** Partition en *Finale* fait avec l'importation d'un fichier *MusicXML* sans édition de ma part.

## 5. QUELQUES FONCTIONS DU SCRIPT.

J'utilise une seule fonction pour tous les appels de fonction du script : *écrire_champ*.

Cette fonction prend comme arguments principaux une définition de la cible d'écriture et les valeurs à écrire.

La cible d'écriture est définie par la fonction *définir_portée*. Elle définit dans quel *instrument* écrire, sur quel niveau écrire (des cinq niveaux de la forme multi-couche : *formes partielles, parties, phrases, cycles, notes)*, et finalement dans quel champ d'objet écrire les valeurs.

Les valeurs à écrire sont normalement fabriquées par une des deux fonctions : *créer_valeurs* ou *obtenir_profils*.

```
(écrire_champ :champ (définir_portée
                      :champ 'proportion
                      :niveau "notes"
                      :nom_dinstrument "fl"
                      :print nil)
              :valeurs (créer_valeurs
                        :nom_dinstrument "fl" :index (index "fl" 0 '() '(1))
                        :champ 'durée
                        :retourner_également 'index
                        :traitements (list
                                      (trai::traitements
                                       :traiteur 'sm:contraste-à-la-même-somme
                                       :args (trai:arg 1.4))
                                      ))
              :valeurs_autrement (obtenir_profils
                                  :nom_dinstrument "fl"
                                  :niveau "cycles"
                                  :champ 'proportion
                                  :traitements
                                  (list
                                   (trai::traitements
                                    :traiteur 'sm:contraste-à-la-même-somme
                                    :args (trai:arg 0.4))
                                   )
                                  )
              :conditions (créer_conditions_composées
                           :évaluer t
                           :prédicat_multi 'and
                           :conditions_simples
                           (list
                            (créer_conditions_simples
                             :valeur_test_0
                             (obtenir_profils
                              :nom_dinstrument "fl"
                              :niveau "notes"
                              :champ 'index_de_cycle_en_phrase)
                             :prédicat_veto 't
                             :prédicat 'member
                             :val_test_1 2)
                            ))
              )
```

**Fig. 12** Un appel de fonction du script.

*créer_valeurs* cherche des données dans *l'analyse* en regardant dans *la forme proportionnelle* tandis que *obtenir_profils* reprend des données déjà écrites dans *la forme proportionnelle*.

Les valeurs à écrire sont normalement fabriquées par une de ces deux fonctions : *créer_valeurs* ou *obtenir_profils*.

*créer_valeurs* cherche des données dans *l'analyse* en regardant dans *la forme proportionnelle* tandis que *obtenir_profils* reprend des données déjà écrites dans *la forme proportionnelle*.

*obtenir_profils* peut aussi filtrer la recherche avec des profils composés des critères.

Les données peuvent enfin être retravaillées par des fonctions de traitement que je peux ajouter à l'argument *: traitements*.

Finalement je peux aussi, à l'argument *: conditions*, imposer des conditions pour que les valeurs soient écrites, et aussi proposer d'autres valeurs à écrire à l'argument *: valeurs_autrement,* si les conditions ne sont pas remplies.

Avec ce petit "kit" de fonctions j'ai pu couvrir toutes les situations qui me sont posées, quand il s'agit de définir une cible d'écriture et des valeurs. Je l'utilise pour tous les paramètres comme le rythme, les hauteurs, les valeurs d'intensités ainsi que les modes de jeu et les articulations.

Finalement, je voudrais aussi faire remarquer que c'est à moi de "fabriquer" les valeurs que je souhaite utiliser. Ici se trouve, je pense, *le point faible* et en même temps *le point fort* de l'environnement ; la plupart du temps j'utilise des fonctions destinées à fabriquer des valeurs pour l'argument *:valeurs*. Or, je pourrais également les fabriquer ailleurs et puis tout simplement coller le résultat ici, ou même taper les valeurs moi-même. Si jamais on a besoin de faire quelque chose un peu hors des sentiers battus, on en a possibilité. Cela représente bien l'état d'esprit de *ejs*, qui est de ne jamais empêcher les *choix directs*[8].

## 6. COMMENTAIRES SUR LE PRATIQUE

*ejs* est ma méthode de composition. C'est un logiciel bâti autour des idées que j'ai voulu réaliser : pouvoir changer et modifier toute une forme globale en gardant la cohérence structurelle. La façon de faire cela est de travailler avec des proportions. Construire et manier une forme de proportions ne se fait pas à la main : les calculs sont tout simples, certes, mais de les réaliser juste pour tester une petite idée qui vient en tête ne se fait pas dans la réalité. En travaillant à la main on fait au maximum quelques essais pour la forme globale et puis il faut que cela soit bon. Ce n'est pas une façon souple de composition : avec seulement quelques essais on ne peut même pas dire que les valeurs sont des *proportions* puisqu'elles sont presque *fixes*. Dans ce contexte j'entends par proportion : élastique, maniable, non fixé, ouvert, passible de changer à tout moment.

La difficulté, de l'autre côté, c'est de ne pas perdre le contrôle - ni le contact - avec toutes ces notes, ces structures. C'est pour cela qu'il faut que je consacre un temps considérable à *apprendre* mes propres structures et à apprendre l'*expression musicale* de ces structures *en les écoutant*.

Je simule *tout le temps*. Je travaille aujourd'hui avec une forme hautement structurée en la façonnant *à l'oreille*. J'insiste sur le fait que la simulation est au centre de mon quotidien. Je sculpte ma composition en l'écoutant. Avec *ejs*, j'ai assuré le niveau de structuration et de cohérence formelle, mais l'oreille est toujours présente : j'écoute, je change et j'écoute encore et je change de nouveau.

Il y a ici un parallèle à faire : cette méthode de composition est très semblable à celle de la composition de la musique électroacoustique. Ce ne sont pas les résultats sonores qui sont comparables mais les procédés, fondés tous les deux sur la validation *à l'écoute* des résultats des traitements algorithmiques d'un matériau musical/sonore pré-existant.

## 7. CONCLUSIONS

Dans les limites de cet article, j'ai présenté le travail logiciel, à travers la description d'un flux de travail et en développant l'exemple d'une construction multicouche.

Dans mon travail futur, je développerai une analyse critique du rendu musical des partitions produites par un tel environnement de travail.

J'insiste encore une fois sur la démarche autonome d'un compositeur-chercheur dont le principal objectif est le développement d'outils logiciels personnels au service de sa création musicale. Selon moi, c'est l'interaction avec la technique de l'écriture musicale, intégrant de nos jours les technologies informatiques toujours en cours de développement (composition avec des moyens informatiques), qu'il s'agit d'approfondir pour servir une esthétique personnelle. D'où, je l'espère, l'intérêt de cette première contribution au moins pour les sciences de l'art.

On ne doit par ailleurs pas sous-estimer l'intérêt d'une telle démarche sur le terrain des sciences, car la question de la relation entre l'homme et ses outils concerne, entre autres, autant l'anthropologie du travail que la musicologie dans l'étude des relations du compositeur avec ses outils ; des domaines qui ne sont pas aussi éloignés de l'informatique musicale, dès lors qu'on aborde celle-ci en dehors du champ des "sciences de l'ingénieur".

## 8. REFERENCES


[1] OpenMusic : Assayag, G., Agon C., Ircam, 1996-2007

[2] Dolet®, Good, M., Recordare LLC, 2001-2008, http://store.recordare.com/dolet4fin.html

[3] MusicXML, Good M., Recordare LLC, California 2001-2007.

[4] Recordare LLC, California, 2001-2007. http://www.recordare.com/

[5] MCL, Digitool, Inc., Lancaster, USA, 1996-2005

[6] AudioSculpt : Hanappe, P., Ircam, 1995-2007


[8] J'entends par *choix direct* une action locale sur la structure sans motivation ou appui structurel. J'utilise le terme dans les deux sens proposés par F. Courtot : l'édition structurale, où un acte d'écriture directe est enregistré et propagé dans un réseau logiciel, et l'édition non-structurale, où les changements manuels restent localement déterminés sans autres conséquences. [Courtot 1993] Cité par [Vaggione 1996] [20].

[7] OXSp : Hedelin, F., Stockholm, 1995-2005

[8] PWGL : Laurson, M., Kuuskankare M., Norilo V., 2002 -2007 Sibelius Academy, Helsinki

[9] Patchwork : Laurson, M., Rueda, C., & Duthen, J., 1993, Paris, Ircam

[10] Finale : Make Music, Inc. 1988-2007

[11] LISP : McCarthy, J., 1958 Massachusetts Institute of Technology (MIT) ; *ANSI CommonLISP :* ANSI X3.226-1994 Information Technology Programming Language Common Lisp, 1994

[12] Elody : Orlarey Y., Grame, Lyon, 1997-2006

[13] Sandgren, J., "jsm-gesture", 1997, Stockholm
http://www.kmh.se/visasida_sv.php?p=4&t=s3,
http://www.joakimsandgren.com/media/files/jsm-gesture.tar.gz

[14] Sandgren, J., *cordes sur bois,* 1997, Stockholm
http://www.joakimsandgren.com/media/scores/cordes%20sur%20bois.pdf, http://www.mediaproduktion.net/Cordes_sur_bois_eng.html

[15] Sandgren, J., *sinfonietta,* edition suecia, Stockholm
http://www.editionsuecia.com/avd/mic/prod/micv5.nsf/AllDocuments/9121E41E7EBFC5E9C125723000768049 ,
http://www.joakimsandgren.com/media/scores/sinfonietta.pdf
http://files.joakimsandgren.com/sinfonietta.mp3

[16] Common Music Notation: Schottstedt B., 1996-2001, CCRMA, Stanford University

[17] Sutton, R., graph, Alberta, Canada
http://www.cs.ualberta.ca/~sutton/index.html
http://www-anw.cs.umass.edu/~rich/graph.html

[18] Sibelius, Sibelius Software Ltd, 1993-2007, Londre

[19] Common Music : Taube R., 1998 - 2007, University of Illinois

[20] Vaggione, H., "Vers une approche transformationnelle en C.A.O", *Actes des Journées d'Informatique Musicale 1996 ,* 1996