Éléments théoriques pour saisir le code de la computer music en analyse musicale

Maxence Larrieu

ÉLÉMENTS THÉORIQUES POUR SAISIR LE CODE DE LA COMPUTER MUSIC EN ANALYSE MUSICALE

Maxence Larrieu
LISAA, Université Gustave Eiffel
maxence@larri.eu
cv.archives-ouvertes.fr/maxence-larrieu

RÉSUMÉ

La musique qui nous intéresse ici a ceci de particulier qu'elle se compose avec du code informatique. Ce dernier, s'il est compris par son concepteur, pose néanmoins des difficultés pour l'analyse musicale en ce qu'il s'avère, de prime abord, abscons. Constitué d'un amas de lignes ou de boites interconnectées il reste *a priori* difficile de faire les liens entre code et musique. C'est dans cette perspective d'analyse musicale que nous apportons ici une étude du code de la computer music. Dans la première partie nous détaillons ses différents aspects : il contient un grand volume d'information ; cristallise des connaissances ; est enchevêtré dans un langage de programmation ; est le fruit d'une activité de design ; possède une architecture ; intègre une sémantique. Dans la seconde partie nous apportons une organisation de ces différents aspects, puis nous donnons un exemple d'utilisation en analyse musicale. Enfin, nous concluons sur l'importance de traverser différents niveaux technologiques pour atteindre les significations musicales portées par le code.

1. INTRODUCTION

Le champ de l'analyse de la computer music a été marqué en 2020 par la publication de deux ouvrages : *Inside Computer Music* (2020) de Michael Clarke, Frederic Dufeu et Peter Manning, et *Between the tracks* dirigé par Miller Puckette et Kerry Hagan (2020). Dans ceux-ci nous trouvons des analyses effectuées en utilisant le code des pièces musicales. Cet intérêt pour le code en analyse, ou plus largement en musicologie, nous montre que l'époque où ce dernier était envisagé comme des « données opératoires » [18] uniquement utiles pour les ingénieurs ou techniciens, semble maintenant bien derrière nous.

Cette attention pour le code nous invite à nous questionner sur son utilisation en analyse musicale. Que les analyses augmentent ne signifie pas en effet que les questions sous-jacentes à cette pratique soient résolues. L'interrogation globale qui nous occupe dans cet article émerge d'elle-même quand nous mettons en regard une partition et le code d'une pièce : comment l'analyse musicale peut-elle s'aider du code des pièces musicales ?

Il s'agit ici d'une question globale qui nécessite au moins des compétences en sciences cognitives, musicologie, analyse musicale, ingénierie, programmation et informatique. L'objet de cet article n'est ainsi pas de répondre à la question mais d'apporter un soubassement théorique, lequel pourra être utilisé dans un second temps pour apporter des éléments de réponse. L'apport théorique est amené en étudiant le code, en éclairant ce qui fait, dans un contexte d'analyse musicale, sa particularité. Dans la première partie nous expliquons différents aspects du code qui nous paraissent primordiaux dans une finalité d'analyse musicale. Nous verrons que le code (i) contient une imposante quantité d'information, (ii) intègre des connaissances sur la production sonore, (iii) est enchevêtré dans un ou plusieurs langages de programmation (iv) est propre à son concepteur (v) possède une architecture et enfin (vi) intègre une sémantique. Dans la seconde partie nous proposons une organisation de ces différents aspects et terminons avec un exemple en analyse musicale.

2. LES DIFFÉRENTS ASPECTS DU CODE

2.1. Quantité d'information : changement d'échelle

Le changement d'échelle apporté par le numérique est présent dès la genèse de l'informatique musicale. En 1963, dans un article présentant la technique de synthèse sonore – *The digital computer as a musical instrument* –, le pionnier Max Mathews relève ainsi un problème de taille : la nouvelle représentation du signal audio que permet le medium numérique possède une quantité d'information qui dépasse l'entendement : *To specify individually 10 000 to 30 000 numbers for each second of music is inconceivable* [9].

L'usage d'un langage de programmation peut ainsi être vu comme un moyen pour s'extraire de cette quantité d'information démesurée (voir également [11]).

Cependant, même s'il permet de s'extraire de l'information démesurée des échantillons audio, l'utilisation d'un langage engendre une quantité d'information importante. C'est flagrant par exemple lorsque l'on met en regard, naïvement, une partition traditionnelle avec le code informatique d'une pièce musicale. Le code est constitué d'un amas considérable de lignes alphanumériques et/ou de boîtes graphiques, qui s'interconnectent et s'encapsulent autant que souhaité. Ainsi dans la pièce pour alto et électronique *Partita I* (2006) de Philippe Manoury le musicologue Frédérique Dufeu [3] a relevé pas moins de 40 000 « objets Max » dans le code de la pièce[1]. De la même manière dans la pièce *Jupiter* (1987) pour flûte et électronique du même compositeur nous avons relevé plus de 8 000 lignes alphanumériques [5] pour un composant du code, communément appelé les QLIST[2].

Cette quantité d'information peut s'expliquer de différentes façons. Relevons d'une part les règles formelles inhérentes aux langages qui engendrent des informations supplémentaires, et d'autre part les dimensions sonores et musicales sur lesquelles le code permet d'agir. Les travaux de Horacio Vaggione nous montrent qu'une des particularités du code est qu'il permet d'écrire des phénomènes sonores à des temps infinitésimaux, ou, pour le dire avec les termes du compositeur, d'agir sur le *microtemps* [20]. La synthèse granulaire s'effectue ainsi avec des grains de quelques dizaines de millisecondes. Composer, agir, sur de telles dimensions appelle nécessairement un changement d'échelle en quantité d'information.

Cette possibilité de composer en « deçà de la note » se retrouve de façon plus aigüe chez Jean-Claude Risset. Dans l'important travail de sensibilisation effectué par le chercheur et compositeur nous retrouvons un idiome récurrent pour qualifier le code : « partition exhaustive », « partition intégrale » [13] [15], comme pour nous alerter sur le fait que le code, la « partition numérique » (nos termes) possède quelque chose de particulier. Ces termes se comprennent en effet par analogie avec la situation instrumentale, que nous pouvons ainsi éclairer : si la partition traditionnelle requiert un instrumentiste pour être « rendue sonore », la partition numérique, le code, lui, n'en a pas besoin. Mais pour permettre cela le compositeur doit décrire exhaustivement la structure sonore souhaitée :

Les programmes stipulant à l'ordinateur la recette de synthèse sont de véritables partitions de la structure sonore, partitions exhaustives et transmissibles [13]

Soulignons enfin que cette compréhension est particulièrement pertinente avec la méthode de synthèse additive, laquelle nécessite une description fine des composantes sinusoïdales à additionner ; méthode de synthèse qui est fortement utilisée par le compositeur, présente par exemple dans *Inharmonique* (1977) et *Songes* (1979), voir [7] et [19].

1 « 40 111 objets Max, répartis dans 1 749 subpatchers », [3]. Il s'agit du patch nommé *PARTITA2011-24.maxpat*

2 Les QLIST sont des instructions alphanumériques écrites par le compositeur qui permettent de décrire dans le temps l'électronique.

2.2. Connaissances

Thor Magnusson effectue depuis plusieurs années des recherches sur les instruments de musique numérique. Dans un article qui a fait date, *Of epistemic tools : Musical instruments as cognitive extension* [8], l'auteur a recours aux sciences cognitives pour éclairer la compréhension que nous avons de ces derniers. Pour cela des comparaisons avec l'instrument traditionnel (« acoustic instrument ») sont effectuées à plusieurs reprises. Un point central apporté se retrouve dans la différence entre des objets faits de matière, qui se présentent au concepteur, comparé à des objets qui sont à construire *ex nihilo* par le concepteur – la fameuse « page blanche » qui se présente à l'ouverture d'un langage de programmation. Du point de vue de la fabrication, l'instrument traditionnel est construit à partir des propriétés des matériaux utilisés, dans une démarche ascendante : *The sound generation (and the required knowledge) is given for free by nature ;* tandis que celui numérique reste à construire, il correspond davantage à une démarche descendante : *in order to make the instrument, we need to know precisely the programming langage, the DSP theory, the synthesis theory, generative algorithms and musical theory, and have a good knowledge of Human Computer Interaction.* [8]. Cette différence[3] amènera l'auteur à qualifier les instruments numériques, particulièrement, d'*outils épistémiques.*

Ces citations nous donnent un premier aperçu des connaissances qui peuvent être présentes dans le code. Dans notre contexte il nous faut nuancer les connaissances en Interaction Homme Machine (IHM). En computer music ces dernières peuvent tout à fait être triviales. Nous pensons par exemple aux éléments graphiques (GUI : Graphic User Unteraction) natifs des langages Max, Pure Data ou FAUST par exemple. Concernant les autres connaissances nous pouvons les organiser en deux sous-groupes, l'un dédié à la production des sons, l'autre à leurs organisations.

2.2.1. Synthèse et traitement sonore

Avec une vue technologique nous pouvons voir le code comme un objet ayant pour finalité, pour fonction, de produire les signaux d'une pièce musicale. Les connaissances pour produire des signaux relèvent de la synthèse sonore ou du sampling pour utiliser des signaux déjà existants. D'autre part, il est rare que les signaux produits ne soient pas traités, modifiés, pour satisfaire la volonté du compositeur : des connaissances en traitement de signal – Digital Signal Processing – sont également à mobiliser.

3 Les correspondances « instrument traditionnel construction ascendante » et « instrument numérique construction descendante » relèvent bien d'une différence et non d'une opposition. Le principe de « page blanche » est en effet mis à mal avec les bibliothèques des langages de programmation.

2.2.2. Algorithme et organisation temporelle

Parallèlement aux connaissances de production de signaux il est d'usage d'en distinguer d'autres, horizontales, relatives à l'organisation temporelle du matériau musical. Ces connaissances sont moins formalisées et formalisables, et nettement plus hétérogènes. Pour l'illustrer ce sous-groupe peut inclure de simples indications de durées, silences, fondus d'entrée ou sortie ; ou bien l'organisation peut être issue d'un algorithme comme dans *Stria* (1977) de John Chowning ; ou encore être réalisée avec des processus aléatoires ; ou enfin regrouper plusieurs de ces moyens d'organisations.

2.3. Du code

L'aspect le plus évident du code, le premier cité par Thor Magnusson, découle de l'usage d'un langage de programmation : le code possède une syntaxe qui est déterminée par le langage de programmation avec lequel il fonctionne. Un langage possède un ensemble de règles formelles qu'il est nécessaire de connaître, de maitriser, afin de pouvoir l'utiliser. C'est ce que Jean-Claude Risset appelle des *conventions formelles* [14].

Du côté de l'analyse, ce codage vient marquer une première difficulté, un premier éloignement. Pour comprendre ce que fait le code l'analyste doit évidemment maitriser les conventions formelles des langages de programmation utilisés. Soulignons ici que les langages de programmation évoluent inévitablement avec la technologie, ce qui appelle une évolution des connaissances des analystes.

2.4. Design

Le terme *design* n'est pas nouveau pour la communauté d'informatique musicale puisqu'il est présent dans la traduction anglaise du métier de « réalisateur en informatique musicale » : Computer Music Designer [22]. Dans notre contexte nous l'utilisons pour souligner que le code est le fruit d'une activité originale, propre à son concepteur, qui fait appel à des connaissances et de la créativité : *design is the interaction between understanding and creation* [21].

Afin d'éclairer davantage ce que nous ciblons un parallèle avec la partition traditionnelle peut être effectué. Dans le célèbre article de Charle Seeger *Prescriptive and Descriptive Music-Writing* [17], l'auteur emploie l'expression « notation conventionnelle » pour qualifier la partition traditionnelle. Le qualificatif conventionnel peut nous guider dans un jeu de renvois entre la notation et le code. D'un côté la notation est dite conventionnelle car elle contient, par exemple, différents symboles pour représenter la durée des notes, ou encore la portée et le placement vertical pour représenter la hauteur. De l'autre côté, un langage de programmation a pour corrélat un ensemble de conventions formelles qu'il est nécessaire de maitriser. La question qui se dessine est la suivante, si l'une comme l'autre des notations possèdent des conventions pourquoi le code reste-t-il si abscons ? Une différence principale entre ces conventions permet d'apporter une réponse, il s'agit des niveaux, ou réalités, sur lesquels le système de notation permet d'opérer. « portamento, legato, détaché, staccato, spiccato, crescendo ... », de même que la notion de note, sont des signes qui renvoient à des emprises sur le musical, tandis que [osc~], [line~] [random] [pack] [*~] renvoient à des éléments computationnels. Les premiers s'inscrivent dans une culture musicale, forgée par la pratique instrumentale, l'écoute, l'étude solfégique et musicologique ; les seconds s'inscrivent dans un environnement computationnel, où la signification des éléments est régie par ce qu'ils produisent, calculent. Les signes de la partition renvoient à un niveau musical, ceux du code à un matériau technologique.

Nous retrouvons ici un point essentiel : le code ne relève pas directement du musical – « code as material is not musical » [8] – ni même de techniques ou d'ingénierie musicale. La surface du code, ce que nous pouvons lire en premier contact, explorer, est le résultat d'une « transformation » d'éléments musicaux en éléments computationnels : afin d'opérer dans le musical il est nécessaire d'opérer dans le computationnel. C'est précisément cette transformation – terme que nous préférons au terme implémentation, lequel relève du domaine de l'ingénierie et s'avère trop linéaire dans notre contexte – que nous visons. Le design se trouve dans l'interaction entre (i) des connaissances *a minima* en traitement et synthèse sonore (ii) des connaissances en programmation afin (iii) de réaliser une idée musicale.

Le second aspect visé avec ce terme se trouve dans les choix effectués par le compositeur. Ces choix, corroborés par l'utilisation des connaissances, nous montrent que le code est fortement singulier à son designer. Dans cette direction le chercheur Otto Laske [6] voit l'ordinateur durant l'acte compositionnel comme son *alter ego*, et le code qui en résulte comme une objectivation de la pensée. D'une façon similaire Thor Magnusson fait un parallèle avec la théorie « The extended mind » des sciences cognitives, laquelle nous invite à voir les objets extérieurs comme des supports faisant partie intégrante de notre cognition. Par prolongement l'auteur nous rappelle ainsi la non neutralité des instruments de musique : ils sont le reflet de choix musicaux : *the piano keyboard 'tells us' that microtonality is of little importance (and much Western music theory has wholly subscribed to that script); the drum-sequencer that 4/4 rhythms and semiquavers are more natural than other types; and the digital audio workstation, through its affordances of copying, pasting and looping, assures us that it is perfectly normal to repeat the same short performance over and over in the same track.*

Ces exemples nous montrent qu'il serait une erreur de considérer le code comme un objet neutre. Il cristallise au contraire une compréhension singulière du

musical, il objectivise des choix musicaux, des connaissances en production de signaux audionumériques, et enfin des compétences en programmation. Nous pouvons ainsi voir le code comme le résultat d'un acte créatif, à la fois musicalement et technologiquement : le code est le reflet de son designer, il cristallise ses choix, ses compétences, ses idées.

2.5. Abstraction : Architecture

Une des principales spécificités des langages de programmation est leurs capacités abstractives, c'est-à-dire la possibilité pour l'utilisateur d'avoir recours à des outils lui permettant d'atteindre des niveaux de significations, de contrôles, plus élevés. Ainsi la première des six dimensions cognitives[4] des langages de programmation présentées par Yann Orlarey [11] concerne l'abstraction : *La capacité d'abstraction, c'est la capacité à généraliser. Abstraction (*abs-trahere *en latin), c'est séparer en quelque sorte l'essentiel du détail.*

En informatique musicale l'abstraction est bien connue puisque les langages Max, PureData et FAUST utilisent le terme même pour nommer cette possibilité[5]. Dans les autres langages les termes varient, relevons *subroutines* pour Csound ou encore *functions* pour SuperCollider par exemple.

L'importance de cette dimension cognitive nous montre, par corrélation, que le code, ce qui qui résulte de l'usage des langages, n'est pas quelque chose de plat, mais qu'il possède une certaine profondeur, laquelle se structure. C'est précisément cette « structuration en profondeur » que nous ciblons ici avec l'architecture du code. Cet aspect du code apparaît nettement dans un travail réalisé par Frédéric Dufeu [3] portant sur les patchs Max des œuvres du cycle *sonus ex machina* de Philippe Manoury. L'idée de l'auteur est d'avoir, pour un premier contact avec le code, un outil technologique permettant de comparer automatiquement les patchs des pièces du cycle afin d'estimer leurs densités et leurs similarités.

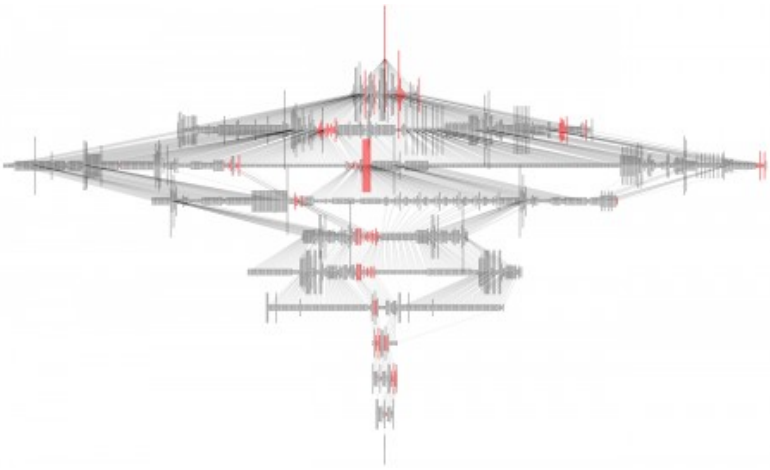

Figure 1. Représentation d'une architecture du patch de *Partita I* (2006), réalisée en fonction des sous-patchs (en rouge apparaissent ceux avec théoriquement un objet audio). Voir la méthodologie dans [3].

L'architecture du code vient corroborer l'activité de design vue précédemment. Les encapsulations présentes dans le code ne sont pas le fruit de connaissances partagées par des communautés, mais sont propres au travail de design, d'un agent. De fait, si nous demandons à deux agents de réaliser le code d'une même pièce à partir des mêmes instructions, nous pouvons dire qu'il n'y a aucune chance pour que nous obtenions les mêmes architectures.

Nous avons vu précédemment que le code, du fait de son inscription dans un environnement computationnel, ne relevait pas directement d'un niveau musical. Nous pouvons à présent augmenter cette remarque : la structure du code ne reflète pas la structure de la pièce.

Finalement, nous pouvons dire que seule la volonté organisationnelle du compositeur régit la structuration du code, son architecture. Dès lors, un artiste, un compositeur, peut très bien faire le choix de rendre son code illisible sans que cela ne se reflète dans la pièce produite. Nous retrouvons cette possibilité dans la thèse d'Alex McLean traitant du *live coding* [10], où l'appellation « architectes astronautes » vient qualifier des programmeurs qui codent à des hauts niveaux d'abstraction afin de brouiller la lisibilité. Dans la même lignée les patchs issus des langages comme Max ou PureData sont connus pour devenir facilement illisibles, comme le montre la figure suivante.

4 Voir aussi les 14 dimensions dans [4] où l'abstraction est également la première dimension définie : *An abstraction is a grouping of elements to be treated as one entity, whether just for convenience or to change the conceptual structure.* Définition que nous n'avons pas retenue car elle ne met pas suffisamment l'accent sur la réduction d'information que permet l'abstraction, sur ce processus par lequel des éléments sont mis en avant au détriment d'autres avec lesquels ils formaient un tout.

5 « *To make an abstraction, save a patch with a name such as "abstraction1.pd" and then invoke it as "abstraction1" in an object box »* peut-on lire dans la documentation de PureData [12]

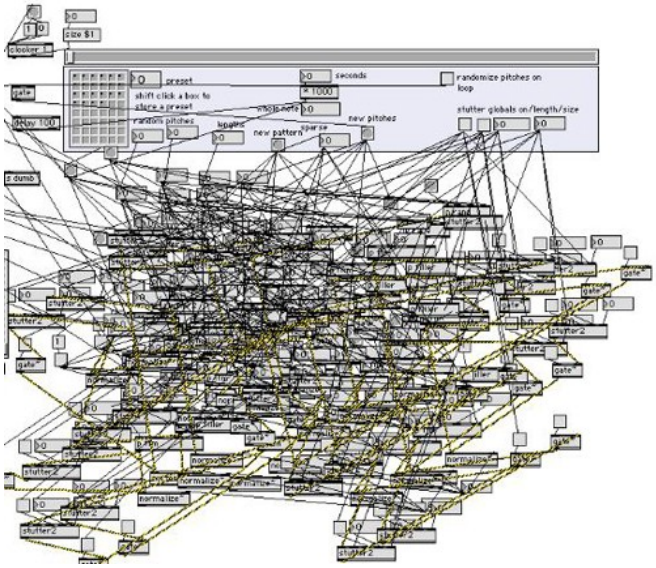

Figure 2. Exemple d'un *patch* Max devenu illisible. Extrait de [1]

2.6. Sémantique

La sémantique est une notion clé en informatique. Avec la syntaxe elles forment deux composantes essentielles dans la conception des langages de programmation. Par exemple dans la documentation de PureData la sémantique fait l'objet d'un chapitre dans lequel sont expliquées les différences entre objets et messages, les différentes classes de messages, les « inlets » et enfin le sens du signe dollar ($). La sémantique que nous visons ici est différente de cette notion informatique en ce que notre objet d'étude n'est pas les langages de programmation mais bien le code, le résultat de l'usage de ces derniers dans le cadre d'une création musicale. La sémantique que nous visons doit se comprendre dans la relation du code et de l'homme. Elle concerne le sens, dans l'interaction avec le compositeur, des éléments du code.

Afin d'amener cet aspect prenons comme exemple un fragment du code de la pièce *Inharmoniques* (1977) de Jean-Claude Risset, utilisé au début de la section IV. Ce fragment permet de simuler des sons de cloches.

Dans ce code il y a d'abord une partie audio (ou « instrument » dans la terminologie de Music V) qui est au plus proche de la production du signal. La figure suivante représente le code réalisé.

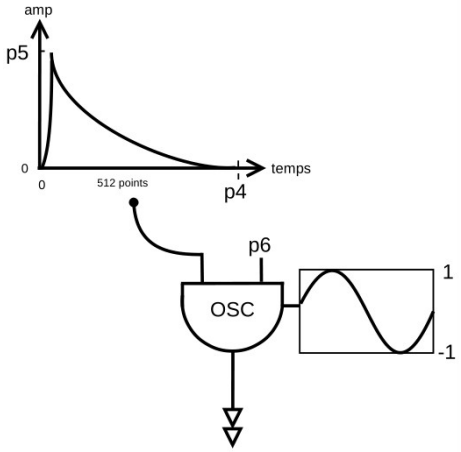

Figure 3. Représentation de l'instrument utilisé pour simuler les sons de cloches

L'instrument possède trois paramètres standards qui permettent de décrire l'évolution d'une sinusoïde : sa durée (P4) son amplitude (P5) et sa fréquence (P6).

Le code contient également une structure de donnée, qui va permettre au compositeur de décrire le son de cloche souhaité globalement, sans passer par la description de chacune des composantes. Cette structure de cloche précise le nombre de composante, l'amplitude globale, la fréquence de référence, le numéro d'instrument, et enfin pour chaque composant un triplet de données (fréquence, durée, amplitude).

Nombre partiels	Amplitue global	Fréquence de ref.	Num. instru	Triplet pour chaque partiel freq, durée, amplitude
9	975	349	3	675, 24, 200
				124, 16, 200
				346, 22, 200
				1108, 17, 200
				1634, 6, 200
				2249, 3, 200
				2941, 2, 200
				3700, 1.5, 200
				4504, 0.8, 200

Figure 4. Structure de données utilisée pour simuler des sons de cloches

Le code contient également une dimension temporelle, appelée « partition » dans ce langage, où le compositeur va modifier dans le temps les valeurs des contrôles. La première séquence se compose des trois lignes suivantes :

```
INS 0 1 ; END ;
PLF 0 6 1 1280 ;
NOT 1 1 0 1000 440 ;
```

Les signes INS END NOT renvoient directement à des conventions du langage. Par contre PLF renvoie à une sous-routine écrite par le compositeur. Elle va permettre de faire le lien entre les événements NOT du code (3e ligne) et la structure de cloche. À défaut de

connaître leur contenu Denis Lorrain [7] en donne une explication littérale. En substance, cette sous-routine permet d'attribuer une durée à chacun des partiels en fonction de la fréquence globale (P6) indiquée et des données de la structure de cloche 1280.

Le dernier aspect du code que nous souhaitons amener se trouve dans la dernière ligne de code la partition « NOT 1 1 0 1000 440 ». Si les trois premiers éléments sont propres au langage, les autres eux sont propres à la composition, au travail de design effectué. Pour les comprendre il faut donc les relier aux autres éléments du code. 0 1000 440 va être redirigé vers la PLF 6 laquelle va actualiser la structure de cloche n° 1280 avec une amplitude de 1000, une fréquence centrale de 440 Hz avec la durée des composants indiqués dans la structure. L'analyste doit donc relier les différents éléments du code afin de comprendre l'action du compositeur. Ensuite, dans son travail d'analyse, il pourra rendre compte des significations acquises :

Les énoncés NOT eux-mêmes spécifient une amplitude (P5) et une fréquence (P6) : la structure sera transposée en amplitude et en fréquence d'après ces paramètres [7].

Cet exemple nous montre que le code intègre des significations qui sont construites par le compositeur. C'est durant l'activité de design, par le recouvrement de significations, que les significations de plus haut niveau, là où le compositeur agît, se réalisent.

Nous voyons ainsi que le code contient une sémantique, un niveau porteur de sens, où le compositeur développe son matériau.

3. RECUL

3.1. Une organisation

Les différents aspects amenés ne relèvent pas tous du même niveau d'abstraction. Par exemple le codage est d'un niveau moindre que celui du design ou de la sémantique. Ainsi pouvons-nous organiser les différents aspects. Relevons d'une part ceux relevant de la technique, comme les connaissances en synthèse et traitement sonore, et d'autre part ceux relevant de la technologie, comme le codage intrinsèque au code ou encore l'architecture permise avec les abstractions. La sémantique, elle, relève d'un niveau musical. C'est dans la confrontation entre « ce que produisent les éléments du code » et « ce que souhaite le compositeur », par interaction, que jaillissent les significations musicales. La sémantique se trouve à un niveau où les aspects techniques et technologiques ont été abstraits, pour atteindre le sens musical des éléments du code. Nous retrouvons cette importance de la signification du concepteur dans l'article sur les « instruments composés » de Robert Schnell et Marc Battier [16] : *After all it will be always the semantic one associates to a given stream and processing module which counts.*

Pour le compositeur le passage à la sémantique ne peut se faire que par le design, en enchevêtrant les aspects techniques et technologiques puis, en les encapsulant dans des abstractions, lesquelles permettent de prendre de la distance vis-à-vis d'une quantité d'information trop importante et surtout d'accueillir des significations musicales, liées aux idées compositionnelles.

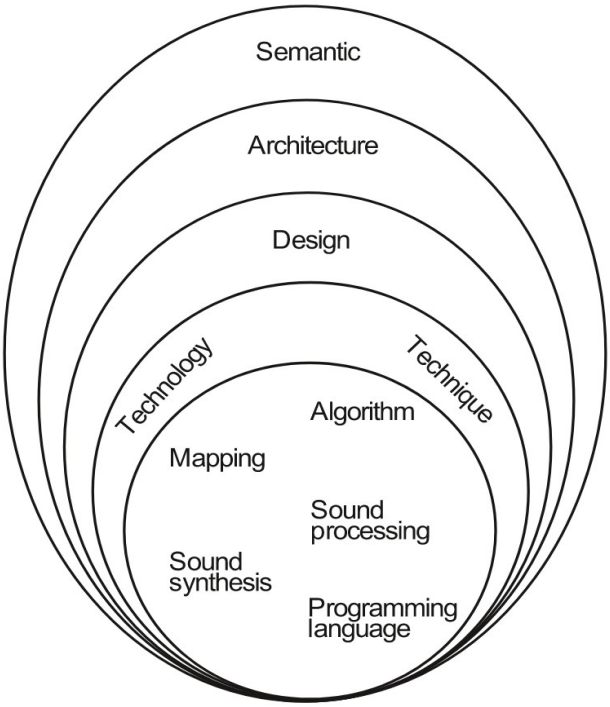

Figure 5. Organisation des différents aspects du code.

La figure précédente représente cette organisation. A un bas niveau se trouve le langage de programmation et les éléments techniques, comme les méthodes de production sonore. De l'autre côté au plus haut niveau se trouve la sémantique, niveau où réside les significations musicales. Entre les deux réside des niveaux plus flous, là où le compositeur se forge ses outils techniques lui permettant de créer et développer son matériau.

3.2. Analyse musicale et code informatique

Cette organisation peut aider pour l'analyse. Elle montre que l'utilisation du code doit dépasser des niveaux technologiques et techniques. Nous l'avons vu à plusieurs reprises, le code, du fait de son encodage, son architecture et de sa finalité computationnelle, n'est *a priori* pas musical. Pour autant la sémantique, qui est à la fois construite et recouverte par le code, relève bien du musical. Pour éclairer une dernière fois la sémantique du point de vue de l'analyse, elle correspond au moment où l'analyste a effectué les liens entre musique et code, au moment où il sait modifier le code pour modifier la musique produite. Pour atteindre ce niveau il est nécessaire de se confronter aux niveaux

technologiques et techniques du code afin de pouvoir les dépasser. Ce dépassement en analyse musicale, ou dans l'étude d'une pièce visant sa reconstruction[6] par exemple, peut poser problème. Nous donnons ci-après un exemple pour l'illustrer.

Le compositeur et chercheur Georg Hajdu a réalisé dans les années 2000 une reconstruction de la pièce *Studie II* (1954) de Karlheinz Stockhausen en utilisant le langage Max[7]. Le dispositif de cette pièce, intégralement analogique, consiste pour l'essentiel de deux modules. Le premier, de synthèse, est un générateur de sinusoïdes et le second, de traitement, est une réverbération : une chambre d'écho de l'ordre de 10 secondes. Une particularité de *Studie II* se trouve dans l'utilisation de la réverbération, puisqu'elle n'est pas utilisée pour traiter, prolonger, un signal déjà diffusé, mais, comme on peut le lire dans les instructions de la partition et comme on peut l'entendre, pour créer directement le matériau de la pièce : les sons purs des sinusoïdes sont envoyés dans la réverbération et c'est le résultat sonore de la réverbération qui est enregistré, le résultat de l'excitation de la chambre d'écho. Le matériau, le son enregistré, est ensuite à développer selon les instructions précises de la partition graphique du compositeur. Le problème de la reconstruction réalisée est que l'utilisation de la réverbération ne correspond pas aux indications du compositeur. Dans la reconstruction la réverbération est utilisée comme un effet ajouté, c'est-à-dire que l'on entend simultanément les sinusoïdes et la réverbération, alors que la composition s'est faite avec le résultat de la réverbération uniquement (voir également [2]).

Dans cet exemple de reconstruction nous voyons que la compréhension de la pièce s'est faite avec des significations techniques – l'opposition entre synthèse et traitement – qui ne correspondent pas à la composition réalisée, puisque dans *Studie II* la réverbération devient finalement matériau musical à part entière. Ces significations techniques, qui émergent dans l'étude du dispositif, auraient dû être recouvertes par des significations de plus haut niveau, musicales, propres à la composition.

4. CONCLUSION

Le cœur de cet article consistait à éclairer différents aspects clés du code dans une démarche d'analyse musicale. Nous avons ainsi relevé (i) la quantité vertigineuse d'information (ii) la cristallisation de connaissances (iii) l'usage de langages de programmation (iv) l'activité de design correspondant à la conception du code (v) la présence d'une architecture et enfin (vi) la présence d'une sémantique. Dans la seconde partie nous avons organisés ces différents aspects. A un premier niveau se trouve le langage de programmation utilisé, lequel détermine l'apparence du code, puis les connaissances en production sonore. A un dernier niveau se trouve la sémantique, l'ensemble des significations musicales avec lesquelles la composition s'est réalisée. Enfin nous avons insisté sur la nécessité, en analyse musicale, de dépasser des niveaux technologiques et techniques. Le code peut être vu comme un enchevêtrement de techniques, technologies et de créativité, qu'il est nécessaire de traverser pour atteindre des significations musicales.

La compréhension du code que nous avons faite s'inscrit dans une volonté, celle de développer l'utilisation du code en analyse musicale. Nous espérons que les éléments apportés puissent permettre de développer cette pratique. En prospective, afin d'avancer dans cette direction nous pensons qu'une théorie pour l'analyse de ces pièces serait opportune. Elle pourrait inclure des outils permettant de favoriser le gain d'abstraction, le dépassement technologique, nécessaire pour l'utilisation du code en analyse musicale.

6 Les questions traitées dans cet article sont communes à la problématique de préservation des pièces d'informatique musicale. Sur ce sujet voir le travail pionnier effectué sur la pièce *Stria* (1977) de John Chowning, dans le numéro dédié du Computer Music Journal : The Reconstruction of Stria, 2017, vol. 31, num. 3, https://direct.mit.edu/comj/issue/31/3 (consulté le 2021-05-01)

7 Voir le site web du chercheur http://georghajdu.de/6-2/studie-ii/ (consulté le 2021-05-01)

5. RÉFÉRENCES

1. Bonardi, A., Barthélemy, J., « Le patch comme document numérique: support de création et de constitution de connaissances pour les arts de la performance ». Colloque International sur le Document Electronique, n. 10, 2007. http://lodel.irevues.inist.fr/cide/index.php?id=298 (consulté le 2021-04-29)

2. Dias, A. S. « Deux contributions à la pédagogie de la musique électroacoustique et de l'informatique musicale », *Journées d'Informatique Musicale,* 2007. hal.archives-ouvertes.fr/hal-03105424/

3. Dufeu, F. « Comment développer des outils généraux pour l'étude desinstruments de musique numériques ? Un prototype en JavaScript pourl'investigation de programmes Max », *Revue Francophone d'Informatique et Musique*, numéro 3, 2013. revues.mshparisnord.fr/rfim/index.php?id=255 (consulté le 2021-04-29).

4. Green, T., Petre, M., « Usability Analysis of Visual Programming Environments: A "Cognitive dimensions" framework », *Journal of Visual Languages and Computing.* vol. 7 n. 2, 1996

5. Larrieu, M. « Analyse des musiques d'informatique, vers une intégration de l'artefact: propositions théoriques et application sur *Jupiter* (1987) de Philippe Manoury. » PhD, Université Paris-Est, Marne-la-Vallée, 2018. hal.archives-ouvertes.fr/tel-01757277

6. Laske, O. « The computer as the artist's alter ego », *Leonardo*, vol. 23, n. 1, 1990.

7. Lorrain, D., « Analyse de la bande magnétique de l'œuvre de Jean-Claude Risset *Inharmonique* », *rapport Ircam*, 26/80, 1980.

8. Magnusson, T. « Of epistemic tools: Musical instruments as cognitive extensions », *Organised Sound,* vol. 14, n. 2, 2009. https://doi.org/10.1017/S1355771809000272

9. Mathews, M. V., « The digital computer as a musical instrument », *Science,* vol. 142, n. 3592, 1963.

10. McLean, C. A., « Artist-Programmers and Programming Languages for the Arts », PhD, University of London, Goldsmiths, 2011.

11. Orlarey, Y., « Entre calcul, programmation et création », in *Le calcul de la musique,* dir. L. Potter, Presse Universitaire de Saint-Étienne, 2009.

12. Puckette, M. « Pd Documentation », *en ligne* http://msp.ucsd.edu/Pd_documentation/, consulté le 2021-05-01.

13. Risset, J-C. « Timbre et synthèse des sons », in *Le timbre métaphore pour la composition,* dir. J-B. Barrière. Christian Bourgeois, Paris, 1991.

14. Risset, J-C. « Calculer le son musical : un nouveau champ de contraintes ? », in *La musique depuis 1945 : matériau, esthétique et perception,* dir. H. Dufourt et J.M. Fauquet. Madraga, Bruxelles, 1996.

15. Risset, J-C. « Problèmes posés par l'analyse d'œuvres musicales dont la réalisation fait appel à l'informatique », in *Analyse et création musicales : actes du troisième congrès Européen d'Analyse Musicale.* L'Harmattan, Paris, 2001.

16. Schnell, N., Battier, M. « Introducing composed instruments, technical and musicological implications », *Proceedings of the 2002 conference on New interfaces for musical expression*, 2002.

17. Seeger, C. « Prescriptive and descriptive music-writing. », *The Musical Quarterly,* vol. 44, n. 2 1958.

18. Stroppa, M. « The anlaysis of electronic music », *Contemporary Music Review,* vol. 1, n. 1., 1984

19. Svisinski, J., Tiffon, V. « Analyse de Songes de Jean-Claude Risset », *ANALYSE – Œuvres commentées du répertoire de l'Ircam,* 2021. https://brahms.ircam.fr/analyses/Songes/ (consulté le 2021-04-29)

20. Vaggione, H. « Son, temps, objet, syntaxe. Vers une approche multi-échelle dans la composition assistée par ordinateur », in *Cahiers de Philosophie du Langage*, n. 3 (Musique, rationalité, Langage. L'harmonie : du monde au matériau), dir. A. Soulez et H. Vaggione. L'Harmattan, Paris, 1998.

21. Winograd, T., Flores, F. *Understanding computers and cognition : A new foundation for design.* Intellect Books, Bristol, 1994.

22. Zattra, L., Donin, N., « A questionnaire-based investigation of the skills and roles of Computer Music Designers. » *Musicae Scientiae* vol. 20, n. 3, 2016. https://doi.org/10.1177%2F1029864915624136