Université Pierre et Marie Curie (Paris VI)

Habilitation à Diriger des Recherches

Langages de programmation pour la composition musicale

par

Carlos Agon

Soutenue devant le jury composé de :

Henry Lieberman	**Rapporteur**
Francois Pachet	**Rapporteur**
Miller Puckette	**Rapporteur**
Gérard Assayag	**Co-directeur**
Jean-Pierre Briot	**Co-directeur**
Jean-Francois Perrot	**Examinateur**
Pierre Cointe	**Examinateur**

Paris mai 7 2004

Remerciements

Je remercie l'Ircam pour m'avoir accueilli pendant l'élaboration de mon travail. Je tiens à remercier en tout premier lieu Gérard Assayag, je ne peux que me réjouir de notre collaboration.

Je remercie Messieurs J.-F. Perrot, F. Pachet J.-P. Briot et Pierre Cointe pour leur confiance tout au long des dernières années.

Je remercie Moreno Andreatta, Jean Carrive, Karim Haddad, Daniel Pressnitzer, Pierre Roy, Bennett Smith et Isabelle Viaud-Delmon, ainsi que Monsieur Daniels et Madame Jameson, sans eux ce rapport n'aurait pas vu le jour.

Un grand merci à Aboud Mohsen et à Jean Bresson, en plus des accents et birgules corrigès, je leur suis reconnaissant de leur agréable compagnie.

Finalement je remercie Karina et Isita.

Avant-propos

Ce travail peut être vu comme un cas d'étude de l'utilisation de l'environnement OpenMusic par un groupe considérable de compositeurs. Les généralisations exposées dans ce texte sont inspirées des rencontres personnelles, tout au long des dix dernières années, ainsi que d'une compilation à paraître sous le titre de « The OM composer's book ». Dans ce livre, nous recueillons divers textes où chaque compositeur présente une de ses œuvres, mettant l'accent sur l'utilisation de l'informatique dans la composition.

Les généralisations que l'on trouvera dans ce texte, indépendamment de leur justesse, peuvent ne pas correspondre à des cas particuliers. Les lecteurs, et plus précisément les compositeurs ici nommés, sont priés de tenir compte de ce fait avant d'entamer des procès pour diffamation ou calomnie.

Trois types de citations bibliographiques sont données dans ce texte : les références traditionnelles ([ABC65]) qui se trouvent dans le chapitre 7. ; les contributions du livre « The OM composer's book » (*[om1]*) qui se trouvent dans l'Annexe A. ; et finalement mes documents (*[1]*).

Table de matières

Introduction

Ma recherche se situe essentiellement dans le domaine de l'informatique musicale. Plus précisément je m'intéresse à la composition assistée par ordinateur (CAO). La CAO concerne principalement l'écriture instrumentale, mais ne s'y limite pas. Ceci explique la grande importance accordée aux paradigmes de la formalisation discrète et du calcul symbolique.

En ce qui concerne l'informatique, la musique représente un domaine passionnant dans la mesure où, très souvent, les problèmes rencontrés sont d'une grande complexité. De même, étant donnée la subjectivité de la musique, les différentes solutions sont soumises à divers points de vue, ce qui implique de concevoir des systèmes ou des modèles en sachant qu'ils devront subir diverses modifications et extensions. Du côté musical, l'informatique apporte en première instance un outil révolutionnaire pour résoudre des problèmes : l'ordinateur. Cet aspect bien que fondamental et attrayant n'est qu'une des multiples modalités de l'aide que l'informatique peut fournir au compositeur. Dans ce texte, nous formulons l'idée que l'informatique peut jouer un rôle dans un acte de création ; la défense de nos propositions se fera bien évidemment en prenant comme cas particulier la composition musicale.

D'une manière plus directe, je m'intéresse aux potentialités de l'utilisation de l'ordinateur pour l'élaboration d'une oeuvre musicale et cherche à la susciter. Je dois pour cela tenir compte du principal concerné : le compositeur. La plus grande partie de ce texte essaie de répondre à la question suivante : comment un informaticien (moi) doit-il concevoir un environnement informatique destiné à un utilisateur (le compositeur) afin de réaliser une tâche spécifique (composer). Convaincu que l'on ne peut donner une réponse définitive à si vaste question, ma proposition vise une reformulation du rôle des divers intervenants. En premier lieu, il faudrait ne pas réduire le statut du compositeur à celui de simple utilisateur (dans le sens informatique courant de ce terme). Ceci est une conséquence directe du fait que l'informatique ne peut pas résoudre le problème de la composition. Toutefois, cette affirmation ne vise pas à mettre en question le pouvoir d'expression de la pensée informatique, elle est plutôt une critique des termes « résoudre » et « problème ». En effet, l'acte de composition n'est pas un problème bien formulé. Toute tentative informatique pour faire la mimesis de la composition doit passer avant tout par une formalisation de l'objet à imiter. Il ne s'agit donc pas de faire un modèle informatique de compositeur, mais plutôt d'étudier la composition et de mieux comprendre la place où le calcul et les concepts informatiques peuvent jouer un rôle pertinent au niveau musical. Nous proposons donc une mutation du paradigme problème/solution vers une approche idée/exploration où les outils informatiques vont permettre soit de réaliser, soit de modifier les intentions du compositeur en fonction des possibilités qui lui sont offertes.

Dans ce sens, un premier pas qui m'est apparu logique a été de proposer directement aux compositeurs des langages de programmation plutôt que des applications pour résoudre des problèmes particuliers. Bien que les frontières entre une application et un langage de programmation ne soient pas bien définies, un outil qui en plus de faire permet de faire faire [Ass93] nous semble plus adapté dans ce cas. Quelque soit l'étiquette de notre environnement informatique, il doit permettre à l'utilisateur de construire des situations autres que celles qui lui sont proposées. S'il existait des différences entre un *end-user* et un programmeur, l'une des premières serait le vécu de leur activité ; notre compositeur-programmeur ne s'intéresse pas uniquement au résultat mais aussi à la manière de le produire. Le terme utilisateur ne doit pas être pris de manière péjorative, bien au contraire il peut être vu comme un idéal difficilement atteignable. Pour quelqu'un qui envoie un e-mail le problème n'est pas le protocole de communication, mais plutôt ce qu'il va écrire.

Bien évidemment, il ne s'agit pas de proposer un langage de programmation et de se dire « voilà, on peut tout faire ». Si le seul critère était le pouvoir d'expression un *80x86 assembler* ferait l'affaire. Plusieurs considérations doivent être prises en compte avant de proposer un langage de programmation aux compositeurs. En premier lieu, un effort doit être fait sur la facilité d'utilisation. Cela implique la définition d'une syntaxe intuitive ainsi que des interfaces conviviales pour l'édition de programmes. Il est à noter qu'il ne s'agit pas de rendre simple ce qui est compliqué : dans aucun cas, un effort dans la syntaxe ne doit ramener à une diminution du pouvoir d'expression. Pour les mêmes raisons de simplicité, un langage pour les compositeurs doit fournir des primitives pour les diverses représentations, internes et externes, des structures musicales, qui seront construites et transformées à l'aide des programmes. Puisque le pari d'un langage de programmation pour un non-programmeur reste de nos jours audacieux, il est nécessaire que le compositeur puisse trouver une place de départ entre la simple utilisation d'une interface et la programmation. Mais quel que soit ce point d'interaction, le langage doit être conçu pour ne pas décevoir l'utilisateur dans une tentative d'augmentation du pouvoir d'expression. Dans le meilleur des cas, il est souhaitable que des fonctionnalités proposées puissent être détournées à d'autres fins. Ces deux dernières caractéristiques permettent à mon sens de rendre un environnement plus intelligent.

L'utilisation d'un langage de programmation oblige le musicien à réfléchir sur les processus même de formalisation et lui évite de considérer l'ordinateur comme une boîte noire qui lui impose ses choix. Les langages de programmation offrent au compositeur une énorme liberté de décision en échange d'un effort dans la formulation et la conception. Mais l'utilisation d'un style de programmation déterminé peut susciter des expressions qui ne seraient pas favorisées dans un autre. Le choix d'un langage de programmation joue un rôle dans la formalisation d'une idée musicale. Ceci nous conduit à réfléchir aussi aux différents modèles de programmation existants. Nous décrirons ici les implémentations et utilisations pour la composition de trois paradigmes de programmation : fonctionnelle, par objets et par contraintes. Etant donnée la diversité des situations musicales où la programmation entre en jeu, un paradigme peut s'avérer plus adapté qu'un autre pour un problème particulier. Nous analyserons pour chaque style de programmation quels sont les contextes musicaux où il est le plus approprié. Bien sûr, il ne s'agit pas de proposer trois langages de programmation différents. Il ne s'agit pas non plus de définir une syntaxe multi-paradigme avec diverses primitives : les langages de ce type ne vont en général pas plus loin que la réunion des parties, et restent donc assez lourds à utiliser. La proposition présentée ici consiste à définir un langage graphique permettant la combinaison de ces paradigmes.

Nous soulignerons tout au long de ce texte l'importance de l'aspect visuel du langage, sans toutefois la traiter explicitement. Mis à part son rôle fédérateur, le langage visuel sert aussi d'interface d'utilisation. Ce langage visuel a été fortement inspiré du projet *OpenDoc* [OrE96] d'Apple. *OpenDoc* proposait l'intégration des composants, appelés *parts*, autour d'un élément fédérateur : *le document*. Avec cette stratégie, *OpenDoc* voulait se défaire de la notion d'application et la remplacer par celle de document ; l'argument étant que le document est un espace de travail plus intuitif pour l'utilisateur. Avec le document, cependant, l'interactivité restait locale aux *parts* (en effet, la mise en relation de deux *parts*, par *scriptage*, relevait de la compétence d'un programmeur expert). L'originalité de mon projet dans ce domaine est d'avoir remplacé le document par une notion plus générale en informatique : le programme *[31]*. Des composants tels que les boutons, les menus ou les textes, qui sont courants pour les utilisateurs, font partie de notre langage, mais en lieu d'apparaître dans des dialogues modaux, ils sont placés dans des algorithmes visuels. Un autre rôle prépondérant des programmes visuels concerne la notation musicale. En effet, les algorithmes visuels sont de bons candidats pour noter des processus compositionnels au même titre que la notation musicale note les résultats de ces processus. Un programme visuel implémentant un processus compositionnel donne une représentation des idées du compositeur dans la mesure où ses composants logiques sont visualisés.

Basé sur les idées précédentes, un environnement de composition appelé OpenMusic (OM) a été implémenté sur le langage CommonLisp/CLOS et tourne sur Macintosh, (deux prototypes avancés existent également pour Linux et Windows). Mais plutôt que de décrire OM (une description détaillée pourra être trouvée dans *[28]* , *[27]* , *[24]*), ce texte sera basé sur son utilisation par des compositeurs, afin d'analyser l'utilisation de langages de programmation pour la composition. Les principaux avantages, ainsi que les limites de cette approche, seront présentés.

Ce document est divisé en cinq chapitres. Le chapitre 1 traite de l'évolution des problèmes auxquels je suis confronté depuis le début de mes recherches : les besoin des compositeurs et leur manière de s'approprier l'outil informatique. En effet, le rôle de l'ordinateur a changé : de simple calculateur il est devenu un outil d'exploration où le compositeur retrouve une place de créateur à part entière devant des possibilités inédites, qu'il pourra maîtriser selon ses exigences. Le chapitre 2 montre comment le compositeur, à l'aide des langages de programmation, pourra assumer ce rôle par rapport à l'informatique. Le chapitre 3, à l'aide d'exemples concrets, expose la pertinence des divers styles de programmation par rapport à des problèmes donnés. Le chapitre 4 résume les principes et contributions de ma recherche en CAO. Le chapitre 5 est consacré à l'étude des limites de notre approche.

La conclusion pose la question de l'avenir des rôles des programmeurs et des utilisateurs selon notre point de vue. Nous y envisageons les recherches futures à mener dans cette direction.

1. Quelques problèmes en CAO

Ce chapitre illustre la nature et la diversité des projets issus de la collaboration entre informaticiens et compositeurs. L'ordre chronologique est employé ici afin de mettre en évidence l'évolution d'une des notions de base du rapport entre l'informatique et la composition : le problème musical. Nous étudierons successivement différents « problèmes-types» de l'informatique musicale, pour formuler l'hypothèse suivante : le problème compositionnel, s'il existe, n'est pas de la même nature que le problème informatique. D'une manière moins péremptoire, je dirais que cette hypothèse vise à dépasser l'idée qui réduit le rôle de l'informatique à la solution des problèmes proposés par le compositeur. Ce chapitre cherche à montrer que lorsqu'on dit que le compositeur a un problème c'est qu'on a déjà formulé ce problème en termes informatiques. Musicalement parlant, si le compositeur a des difficultés à composer, l'informatique n'y peut rien. Par contre, nous voulons mettre en évidence l'utilité de l'informatique pour la création de situations musicales qui, à leur tour, pourront être formalisées.

1.1 Problèmes utopiques et commandes

Mes recherches en informatique musicale ont commencé vers 1990. Dans mon mémoire de fin d'études d'ingénieur *[34]*, je me suis posé le problème de l'extraction d'une partition à partir de l'enregistrement numérique d'une oeuvre. Cet objectif, sous la direction de Camilo Rueda, a été raisonnablement réduit à l'extraction de la hauteur à partir d'un son harmonique et monophonique de durée déterminée (et très courte). Le résultat final a été l'implémentation de certaines idées extraites de [Mar72]. Cette application fonctionnait pour certains cas seulement. Aujourd'hui, si je devais reprendre mon projet initial, le résultat ne serait certainement pas meilleur. Il y a dans le domaine de l'informatique musicale de nombreux projets reposant sur des idées similaires. Celles-ci sont souvent en avance sur les possibilités offertes par l'informatique. De plus, les problèmes de ce type ne sont généralement pas clairement définis et leurs difficultés ne semblent pas être bien maîtrisées.

A l'opposé, il existe des problèmes clairement spécifiés, souvent techniques, mais qui ne favorisent pas la collaboration entre compositeurs et informaticiens. Ma première « commande » à l'Ircam est un bon exemple. Mon projet consistait à implémenter des opérations arithmétiques pour des flottants sur une carte DSP. Longtemps après, j'ai su que mon travail concernait l'élaboration d'un *driver* pour le synthétiseur Chant *[33]*. Du côté des compositeurs, je suppose qu'ils étaient peu intéressés par la syntaxe du langage de la carte DSP. Pour eux, il fallait avant tout que le synthétiseur marche.

Dans ces deux premières activités, il y a deux types de problèmes que je cherche à éviter : l'imposition d'outils par l'informaticien et l'exécution stricte d'une commande pour le compositeur. Nous verrons par la suite d'autres problèmes, à mon sens, plus intéressants.

1.2 Un problème ouvert : la quantification rythmique

Le problème de la quantification rythmique, classique en recherche musicale, est une des occupations principales de l'équipe Représentations Musicales de l'Ircam depuis une dizaine d'années. Il peut se définir comme la conversion d'un flux des durées exprimées par des valeurs réelles en une structure rythmique, hiérarchique et discrète, qui puisse être exprimée de façon significative dans le système usuel de notation musicale. L'intérêt de la question provient du fait que beaucoup de compositeurs utilisent des formalismes numériques pour engendrer des configurations complexes de durées. Le problème se pose aussi de façon évidente dans le cas de la transcription automatique du jeu instrumental. Au cours de mon stage de DEA, et en collaboration avec divers chercheurs et compositeurs, j'ai réalisé un programme de quantification appelé Kant *[32]*. L'idée de base dans Kant était de situer la quantification rythmique bien au-delà du traitement purement mathématique. L'un de nos apports principaux a été la remise en question de la définition d'une « bonne quantification ». En effet, généralement, elle tenait compte uniquement de la fidélité à la réalité. Pour nous, la simplicité de le notation et l'information structurelle livrée par une transcription ont aussi une grande importance. Un autre apport intéressant a été la différenciation entre quantification et transcription. Notre modèle peut prendre place dans le processus compositionnel grâce aux différentes transformations appliquées aux données tout au long de la quantification. Le problème de la quantification reste ouvert de nos jours ; il est d'actualité grâce aux divers travaux menés autour de la description et de la recherche par contenu.

Pourquoi le problème de la quantification rythmique n'est-il pas encore résolu ? Est ce un problème non-calculable ? Le génie qui le résoudrait n'est-il pas encore né ? Ou plus simplement le problème n'est-il pas encore bien formulé ? Si résoudre le problème de la quantification rythmique signifie rentrer des données, faire *enter*, puis récupérer une partition, une solution semble très plausible. En revanche, décider automatiquement si un résultat est une bonne quantification me semble peu probable. Pour ces raisons, nous avons évolué de l'idée d'une quantification automatique vers un système mixte entre automatisme et décision, mis entre les mains d'une personne experte. Le travail de DEA de Benoit Meudic [Meu99] a consisté à étendre les possibilités du logiciel Kant de manière à pouvoir analyser et représenter des structures rythmiques présentant une pulsation et une métrique régulières. Quelques nouvelles méthodes de segmentation ont été proposées, basées en particulier sur le coefficient d'autocorrélation ou bien sur le concept de marquage, visant à reconnaître des structures rythmiques usuelles (de métrique simple et de tempo constant sur plusieurs mesures) et/ou des structures rythmiques plus particulières (comportant des changements de métrique et de tempo).

La notion de marquage a été reprise comme base pour le travail de doctorat de Benoît Meudic [Meu04]. En s'appuyant sur diverses théories, mais notamment sur la théorie du rythme de Pierre Lusson [Lus86], nous avons produit des algorithmes pour l'extraction automatique de la pulsation et de la métrique. Le principe général est de considérer la séquence à segmenter comme un ensemble d'éléments. Chacun des éléments répond à plusieurs propriétés (e.g. des intervalles de hauteurs, d'onsets, de durées et d'intensités). Un poids (marquage) est ensuite attribué à ces éléments ; si certaines de leurs propriétés répondent ou non à un critère déterminé (par exemple, les intervalles suivis d'un intervalle plus petit seront marqués). Finalement,

ces idées ont été appliquées pour la recherche de motifs, nous insérant ainsi par la même dans le domaine du *music information retrieval*. Plus récemment, et dans le même domaine, la thèse de Gilbert Nouno est consacrée à la reconnaissance automatique du tempo. Il utilise des algorithmes incrémentaux, ce qui le place directement dans un cadre d'application en temps réel.

1.3 Un problème bien défini : les CRRCM

Il existe cependant des problèmes musicaux bien définis : la construction d'un Canon Rythmique Régulier Complémentaire est un bon exemple. Un canon rythmique est une polyphonie de rythmes identiques déplacés dans le temps. Tout canon rythmique est donné par le choix d'un rythme de base qu'on appellera R, et d'un rythme externe (formé par les entrées des voix) qu'on appellera S. La Figure 1 montre un exemple de canon rythmique ayant comme rythme de base R=(2 8 2) et comme rythme externe S=(5 1 5 1) :

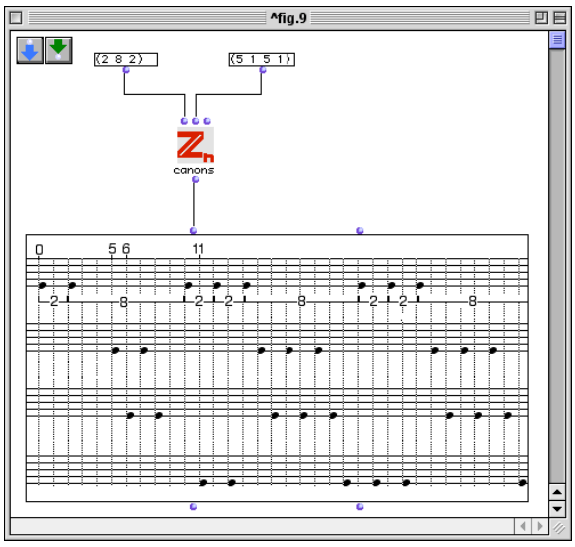


Figure 1. Exemple d'un canon rythmique

On peut vérifier dans notre exemple que dès que la dernière voix est entrée, chaque pulsation est remplie par une attaque de note. Dans ce cas, le canon rythmique est régulier. Nous pouvons aussi observer dans notre canon que jamais deux voix n'attaquent une note à la même pulsation. Les canons vérifiant cette propriété seront appelés complémentaires. D'un point de vue mathématique, un canon rythmique régulier complémentaire correspond à une factorisation d'un groupe cyclique **Z/nZ** comme la somme directe de deux sous-ensembles. Dans le cas du canon précédent, il suffit d'interpréter les structures rythmiques R=(2 8 2) et S=(5 1 5 1) comme ensembles de classes de résidus. On obtient ainsi les deux sous-ensembles R'={0,8,10} et S'={0,5,6,11}. On peut vérifier que les deux sous-ensembles R' et S' représentent une factorisation de **Z/12Z**, dans le sens où chaque élément du groupe cyclique **Z/12Z** peut s'exprimer (de façon unique) comme la somme d'un élément de R' et d'un élément de S'. La Figure 2 montre la représentation circulaire des deux sous-ensembles précédents et du processus de factorisation.

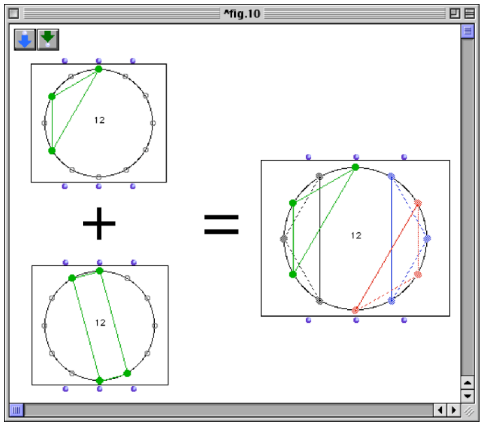


Figure 2. Factorisation du groupe cyclique Z/12Z

Construire des canons tels qu'on les a définis ne présente pas une énorme difficulté du point de vue informatique. En effet, nous avons formulé ce problème sous forme de contraintes *[23]* et nous avons obtenu divers canons (solutions) dans des délais raisonnables. Cependant, nos solutions étaient rythmiquement « triviales » pour les compositeurs, elles étaient toutes constituées de sous-rythmes périodiques. Dans notre exemple, R' est organisé selon une propriété de « transposition limitée » (i.e. des sous-ensembles périodiques), comme on peut le voir dans la Figure 3.

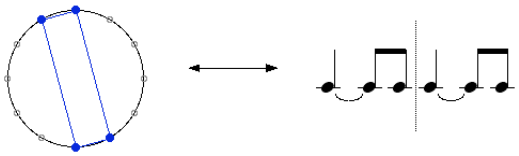


Figure 3. Structure rythmique attachée au mode à transposition limitée (5 1 5 1)

Pour éviter ce type de solutions, nous ajoutons à nos canons la contrainte supplémentaire de n'avoir des modes à transpositions limitées dans aucun des deux facteurs. On obtient ce qu'on appelle des canons réguliers complémentaires de catégorie maximale [Vuz91]. La Figure 4 montre un exemple d'un tel canon rythmique.

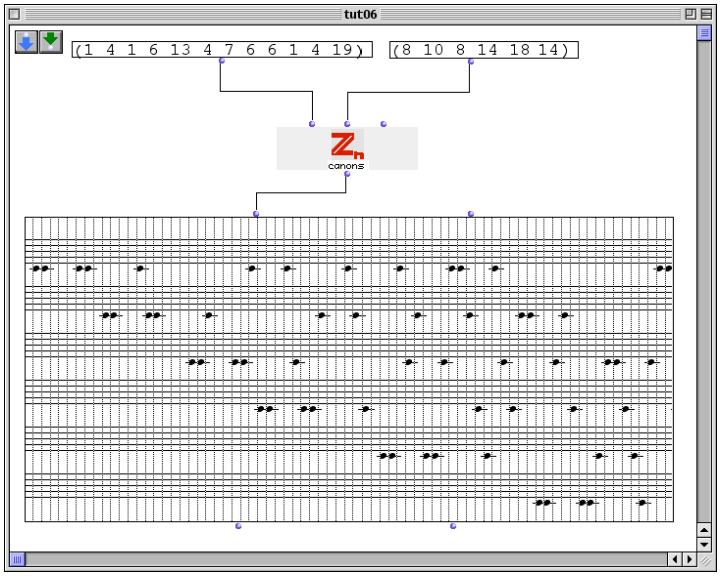


Figure 4. Un canon régulier complémentaire de catégorie maximale

Nous avons implémenté dans OM l'algorithme algébrique proposé par D. T. Vuza comme outil pour caractériser la famille des canons réguliers complémentaires de catégorie maximale et nous avons pu ainsi envisager une étude systématique des solutions possibles pour un groupe cyclique donné. L'algorithme offre d'abord des critères pour établir la liste des périodes des rythmes qui admettent la construction de nos canons. En effet, un tel problème n'admet pas de solutions pour des groupes cycliques d'ordre inférieur à 72, ce qui rend trop lourde, en termes de calcul, une stratégie imposant les contraintes de non-périodicité.

Cette famille de canons rythmiques a eu des développements compositionnels récents, en particulier grâce au travail d'implémentation dans OM et à l'intérêt de nombreux mathématiciens et compositeurs qui ont élargi le modèle original proposé par Vuza, tout en soulevant des questions théoriques qui restent en grande partie ouvertes. Dans *[20]*, nous avons mis au point de nouveaux algorithmes pour calculer des canons où l'identité rythmique est remplacée par d'autres relations d'équivalence. Nous avons en particulier construit des canons où les voix sont équivalentes à une inversion ou à un étirement près. Une formalisation nouvelle des canons rythmiques a été introduite par Andranik Tangian [Tan01]. Elle est basée sur la représentation polynomiale. A partir de cette représentation, Emmanuel Amiot a reformulé la construction d'un canon comme la solution d'un système d'équations de polynômes cyclotomiques *[11]*.

Du point de vue compositionnel, l'un des premiers utilisateurs à s'approprier nos outils a été le compositeur Fabien Lévy. Cependant, pour F. Lévy les notes dans un canon ont été remplacées par des gestes musicaux complexes (glissandi, notes répétées, tremolos, etc.) et des contraintes de continuité ont été imposées entre ces objets musicaux. Les canons résultants, bien que basés sur la grille rythmique obtenue à travers notre algorithme, réalisent une polyphonie de gestes qui masquent l'idée originale du canon. Une autre stratégie compositionnelle a été proposée par Tom Johnson qui trouve la contrainte de catégorie maximale très lourde au niveau musical. En effet, une période de 72 pulsations semble poser des problèmes au niveau perceptif. L'alternative proposée par T. Johnson consiste à générer un canon « toujours » régulier et complémentaire, c'est-à-dire qu'il n'y pas besoin d'attendre l'entrée de la dernière voix pour obtenir le pavage de l'axe temporaire. Mais c'est peut-être le compositeur George Bloch qui s'est intéressé le plus à notre modèle. En effet, pour lui un canon est pris comme une entité sur laquelle il va appliquer des opérations : modulation, compactage de voix et transformation d'un canon en texture. Cette approche fort intéressante est développée dans [And03].

Il nous semble remarquable, dans les trois cas précédents, qu'à partir d'un problème bien défini et de sa solution systématique, le compositeur ait toujours cherché à détourner le modèle théorique pour envisager des applications compositionnelles qui ont radicalement changé la structure initiale. Autrement dit, les structures rythmiques résultantes, bien qu'inspirées du modèle théorique, sont tout sauf des CRRCM.

1.4 Un problème récurrent : La combinatoire

Dans son Harmonie Universelle, Mersenne s'interroge sur la possibilité de composer le meilleur chant imaginable. Après avoir effectué de nombreux catalogues, il se rend à l'évidence de l'impossibilité de son programme à cause de l'énorme nombre de solutions, qui l'empêche de procéder avec une technique d'essais et d'erreurs. On apprend ainsi qu'il existe 362880 mélodies de neuf notes sans répétition ; 90720 si les notes sont choisies parmi sept dont la première

peut être répétée 2 fois, la deuxième 2 fois, et les 5 dernières une fois seulement. Finalement, il arrive au nombre de 324 906 785 280 mélodies si les notes sont choisies dans un réservoir de trois octaves [Ass98]. Il ne semble pas insensé de nos jours d'admettre que, chez le compositeur, calcul et intuition sont présents dans des proportions variables. La systématisation de l'idée combinatoire a débouché naturellement sur l'idée d'automatiser, à l'aide de l'ordinateur, certains aspects de la production musicale. Sans chercher à confier à la machine l'organisation à tous les niveaux d'une architecture musicale, les musiciens construisent avec son aide des matériaux structurés qui peuvent consister en systèmes d'échelles, d'accords, de rythmes, de timbres sonores. Ces systèmes, ou modèles, une fois programmés donnent lieu, par paramétrage, à une exploration systématique.

Un des problèmes combinatoires qui a le plus intéressé les théoriciens de la musique, en particulier au XX^{e} siècle, est celui du classement d'accords. Nous nous sommes intéressés à ce type de problème dans une perspective algébrique visant à établir des catalogues d'accords dans lesquels la notion d'équivalence entre structures musicales dépend de façon étroite du choix d'un groupe opérant sur l'ensemble des notes. Nous avons décrit cette démarche théorique dans *[7]*. L'implémentation de cette approche dans OM nous a permis de rendre variable la structure de groupe la plus appropriée par rapport au type de relation d'équivalence employée. Chaque structure de groupe peut être considérée comme le choix d'un paradigme d'interprétation particulier dans le processus de formalisation des structures musicales. Le groupe cyclique Z/nZ offre une formalisation algébrique du concept d'équivalence entre accords à une transposition près. Ce premier catalogue comprend 352 classes d'équivalences. L'énumération des classes de transpositions d'accords est donc équivalente à l'étude des orbites d'accords par rapport à l'action du groupe Z/nZ sur lui-même. Un premier problème concerne l'énumération de toutes ces orbites pour un tempérament donné. La Figure 5 montre un programme prenant n=12 et n=24. Il y a, par exemple, 80 hexacordes dans la division de l'octave en 12 parties, alors qu'il y en a plus de 5000 dans le système en quarts de ton. Cela donne déjà une bonne idée de la complexité combinatoire engendrée par des grandes valeurs de n.

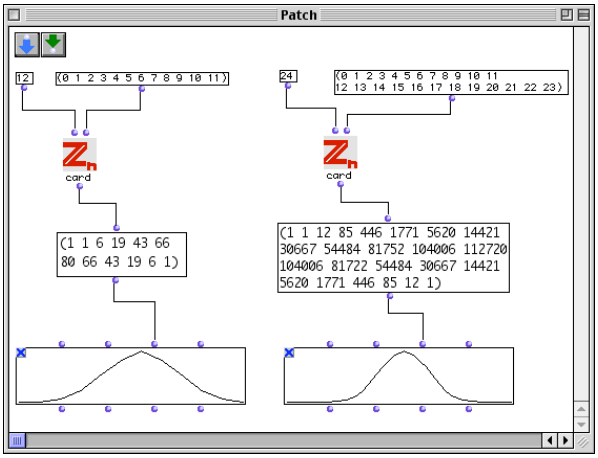


Figure 5. Nombre de classes de transpositions d'accords dans le système tempéré à 12 et à 24 degrés.

En changeant le type de groupe opérant sur l'ensemble des notes, on peut réduire l'espace combinatoire d'accords possibles tout en gardant une pertinence musicale pour le catalogue ainsi obtenu. C'est le cas, par exemple, de l'action du groupe dihédral sur l'ensemble Z/nZ. Cette action détermine les classes d'équivalence d'accords à une transposition et une inversion près. Dans la Figure 6 un accord de do majeur est transformé en accord de do mineur par une

inversion (par rapport à l'axe qui passe par les points 0 et 6) suivie par une transposition (d'une quinte).

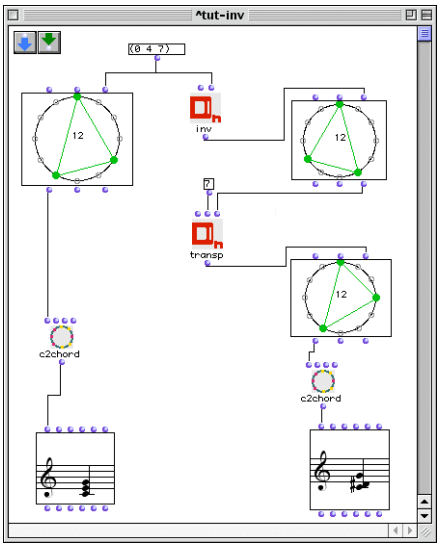


Figure 6. Equivalence majeur/mineur dans le paradigme Dn

De la même manière que pour la transposition, on peut calculer dans ce groupe, pour toute division de l'octave, le nombre d'accords pour une cardinalité déterminée. La Figure 7 offre une énumération comparative pour le système tempéré à 12 et à 24 degrés.

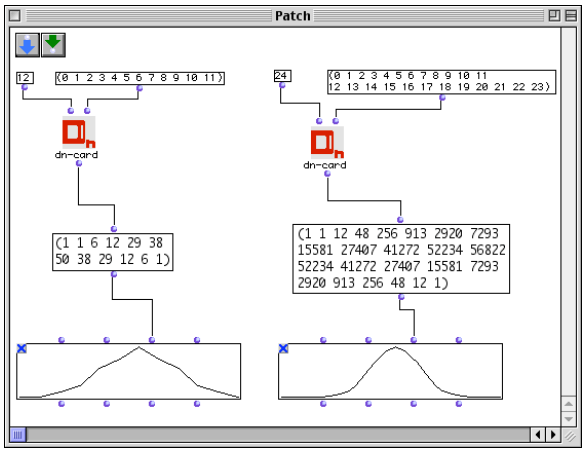


Figure 7. Distribution du nombre d'orbites sous l'action de Dn pour une division de l'octave en 12 et en 24

L'utilisation du groupe affine pour définir la relation d'équivalence entre accords réduit le catalogue des orbites à 88 représentants. Pour comprendre comment le groupe affine opère sur l'ensemble Z/nZ, reprenons le cas de l'accord majeur sous l'action des deux groupes étudiés précédemment. Les Figure 8 et Figure 9 montrent, respectivement, l'action du groupe cyclique et du groupe dihédral.

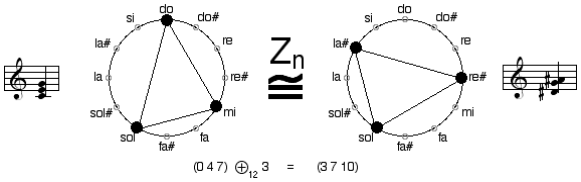


Figure 8. Action « transpositionnelle » du groupe cyclique sur un accord majeur

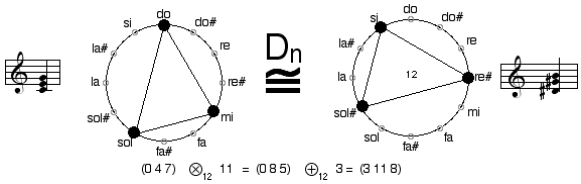


Figure 9. Action du groupe dihédral sur l'accord majeur ou équivalence majeur/mineur

Dans le cas de l'action du groupe affine, tout intervalle de quinte ou de quarte est transformé dans l'intervalle du demi-ton chromatique. La Figure 10 montre la transformation de l'accord majeur sous l'action du groupe affine.

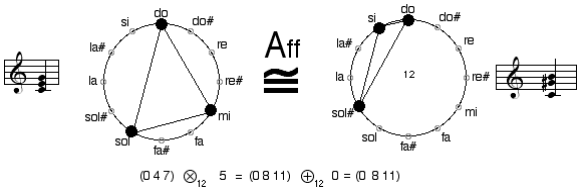


Figure 10. Action du groupe affine sur l'accord majeur

Comme dans le cas du groupe cyclique et du groupe dihédral, l'espace combinatoire issu de l'utilisation du groupe affine possède des caractéristiques de symétrie qui dépendent de la cardinalité des accords considérés. Autrement dit, le nombre d'orbites affines de k et n-k éléments est le même. Cette propriété n'est plus vraie dans le cas d'autres actions de groupes. Par exemple, nous avons implémenté l'action du groupe symétrique (ou groupe de permutations) sur l'ensemble des structures intervalliques. Par rapport aux 43 orbites cycliques, aux 29 dans le paradigme dihédral et aux 21 orbites affines, le catalogue des orbites « permutationnelles » se réduit à 15 représentants. En outre, la distribution des orbites ne garde plus le caractère de symétrie formelle qu'elle possédait dans le cas des trois groupes précédents. La Figure 11 montre la distribution des accords dans ce nouveau catalogue.

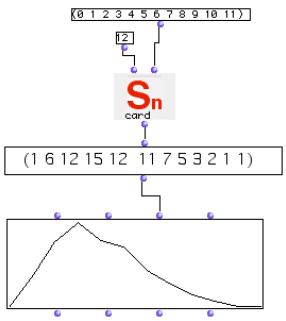


Figure 11. Distribution des orbites dans le cas de l'action du groupe symétrique sur Z/12Z

En conclusion, en introduisant le concept d'action d'un groupe sur un ensemble, toute démarche analytique visant à établir un catalogue exhaustif d'accords peut s'interpréter comme le choix d'un paradigme à l'intérieur duquel on va déduire une notion d'équivalence au sens mathématique. La possibilité de pouvoir changer de paradigme, selon le style à analyser ou la démarche compositionnelle à poursuivre, représente l'un des enjeux majeurs d'une approche computationnelle de type algébrique pour la formalisation des structures musicales. Les quatre groupes, définissant quatre paradigmes d'investigations possibles, assouplissent la notion d'équivalence entre structures musicales, tout en interrogeant la notion d'algorithmes permettant la calculabilité explicite des orbites sous l'action d'un groupe donné *[4]*.

L'apparition de l'ordinateur a suscité un regain d'intérêt pour la combinatoire : les espaces de recherche gigantesques entrevus par Mersenne sont, semble-t-il à portée de puissance des calculateurs. Mais d'une part, et même avec des puissances de calcul non négligeables, la combinatoire musicale à l'échelle d'une pièce entière et non plus de quelques accords reste d'un coût considérable. D'autre part, un problème, qui avait été quelque peu escamoté tant que ces questions n'avaient pas rejoint le champ expérimental, resurgit : l'énumération de solutions est peu de chose si l'on ne dispose pas d'un modèle fixant les critères de choix. On ne sait toujours pas pourquoi devant la multitude des solutions le compositeur, sûr de son métier, va directement à la bonne. Une chose est certaine d'après notre expérience : le praticien ne procède pas de manière énumérative.

1.5 Un problème byzantin : La notation

Construire des structures, exprimer des calculs, n'ont d'utilité que dans la mesure où données et résultats peuvent être interprétés et représentés dans des dimensions musicalement significatives telles que hauteur, durée, intensité ou timbre, pour se limiter aux catégories traditionnelles. Fournir des interfaces visuelles et auditives qui améliorent cette interprétation est alors impératif. Dans la perspective de l'aide à l'écriture, une grande importance est donnée à la notation musicale traditionnelle. Celle-ci agit idéalement avec un logiciel de CAO, non seulement comme matérialisation des informations circulant dans le système, mais aussi comme support de l'inventivité formelle. A ce titre, la notation devrait être dotée de la même souplesse, de la même ouverture (en termes d'extensibilité et de programmabilité) que le langage informatique lui-même et constituer, à terme, le milieu naturel d'expérimentation de l'utilisateur. Les niveaux du langage et de la notation tendront alors à se confondre du point de vue de l'utilisateur. L'écriture elle-même sera prise comme univers de formes, de structures et de relations.

Une fois cet objectif posé, la question de la notation surgit immédiatement. Nous avons proposé une définition dans *[29]* : la notation est un dispositif à deux faces. Une face concrète — un jeu de signes matérialisés (sur la feuille, sur l'écran)— et une face abstraite —un jeu de relations contraignant l'utilisation de ces signes. L'écriture est le processus dynamique d'utilisation de la notation pour mettre en forme des structures et des relations de plus haut niveau. Le premier de ces niveaux supérieurs de structuration est directement discernable dans la partition. Le second niveau de structuration de l'écriture n'est pas directement discernable : il faut pour cela imaginer une exécution virtuelle, ce que font les musiciens expérimentés à la lecture d'une partition. Il y a besoin ici d'une représentation qui aille au-delà de la notation traditionnelle. Nous avons évoqué le cas de figure, traditionnel, de la partition musicale, mais dans le cas qui nous préoccupe, c'est-à-dire avec l'ordinateur, la partition n'est plus qu'un aspect parmi d'autres. En effet, le compositeur, devant son écran, manipule divers objets dont certains

sont des objets de calcul. Cela n'est pas nouveau en soi : il semble évident que la musique ne se compose pas uniquement sur la portée.

C'est alors à tous ces niveaux qu'intervient la notation : notation des programmes calculant des matériaux musicaux ; notation des résultats de ces programmes, sous forme mathématique (des courbes) ou musicale (des notes) ; notation des structures formelles qui articulent ces matériaux ; notation des programmes qui prennent en entrée ces structures formelles et les transforment encore. On entre dans une récursion infinie qu'il appartient au créateur de terminer quand il le décide en calibrant le niveau de complexité qu'il désire (ou qu'il peut) maîtriser.

Dans *[18]*, nous décrivons les divers types d'éditeurs musicaux implémentés dans OM. On peut émettre plusieurs critiques au système de notation musicale traditionnel : le manque d'information sémiotique, le manque de précision (spécialement en ce qui concerne les phénomènes de changement de tempo), l'ambiguïté que représentent les différentes représentations d'un même objet, l'impuissance à exprimer exactement une valeur désirée, etc. Cependant, malgré les problèmes que cela soulève, nous avons choisi pour notre représentation le système de notation musicale traditionnel. En effet, il s'agit là de la façon la plus universelle d'écrire la musique. La création d'un nouveau système de représentation ou la modification du système traditionnel est une tâche hors de notre portée.

Les logiciels de notation musicale dans le domaine informatique continuent à susciter beaucoup d'intérêt de nos jours, mais malgré les divers efforts, nul programme ne s'est imposé de manière consensuelle. Ceci peut s'expliquer par la diversité dans la finalité de leur utilisation : visualisation, gravure, édition et saisie, etc. Comme nous l'explicitons dans *[14]*, nous nous sommes orientés vers ce qu'il conviendrait d'appeler la représentation de la notation musicale symbolique. L'idée principale est d'offrir une solution efficace aux applications dont la finalité est la manipulation et la construction des structures musicales. Notre approche vise à éliminer la prise en considération du formatage final concrétisé par la mise en page en vue de l'édition. Cette stratégie nécessite une méthodologie pour le rendu qui comblera le manque d'information concernant la pagination. Si nos éditeurs ne sont pas très performants au niveau de la gravure, ils présentent diverses propriétés qui nous semblent essentielles à la composition : la visualisation des relations locales entre plusieurs sous-structures de la partition ; un protocole pour la définition d'opérations graphiques visant à modifier la partition ; un accès graphique et une possibilité de navigation à l'intérieur de la structure hiérarchique de la partition. Un exemple de cette dernière caractéristique est montré dans la Figure 12.

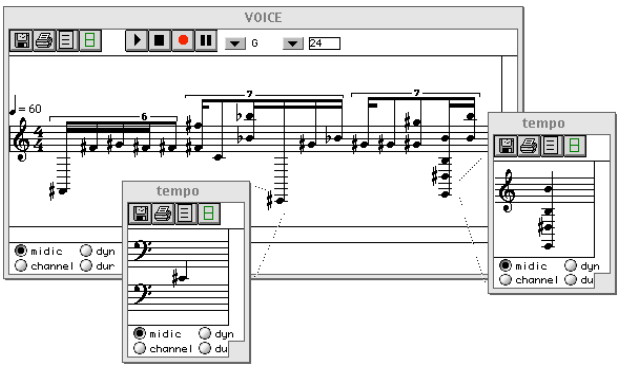


Figure 12. Navigation graphique dans une partition

L'idée principale dans OM est de placer le compositeur dans un environnement purement graphique où l'interaction se situe entre la programmation et la simple utilisation d'une interface. Les éditeurs musicaux sont à la base de cette double interaction. Un éditeur joue un double rôle : représentation d'un résultat dans un arbre de calcul, et composant, au sens d'architecture de composants, d'un document ne présentant que des interfaces d'utilisation. Notre proposition, concernant la notation, est d'intégrer les programmes visuels (*patches*) et les éditeurs de notation musicale. Trois modalités sont proposées *[3]*.

- Inclusion de la notation dans les programmes :
Les programmes visuels sont étendus par un nouveau type de composant appelé *factory*. Une *factory* permet de créer des instances d'une classe musicale donnée (i.e. *voice*, *chord*, *note*). Outre leur rôle dans un calcul, les *factories* peuvent être vues comme des entités autonomes permettant de visualiser et d'éditer leurs instances. Visuellement, le rapport entre instances et éditeurs est inspiré du paradigme *Model-View-Controller* [GHJ95], où le *model* est l'instance musicale, la *view* est sa représentation sur l'écran et le *controller* est une interface permettant de changer les valeurs de l'instance. Les *factories* permettent la visualisation d'une donnée à un moment particulier du calcul ainsi que la mise à disposition d'une « application » indépendante, la partition, permettant l'écriture manuelle de la musique.

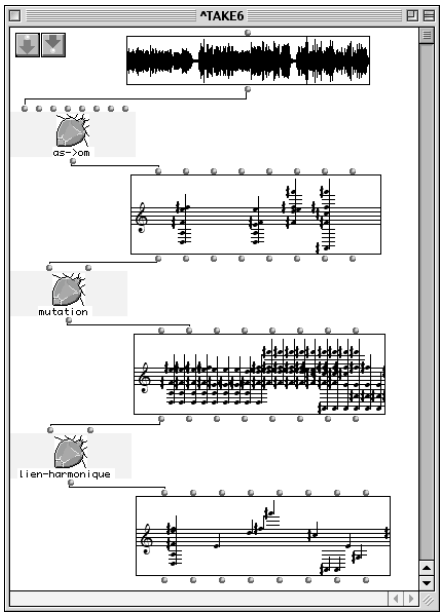


Figure 13. Inclusion de la notation dans un programme visuel

La Figure 13 montre un programme visuel qui construit une mélodie à partir d'un fichier son. Deux types de composants sont distinguables dans ce programme : les boîtes de transformation et les *factories* de sons et des séquences d'accords. L'expérience avec des compositeurs nous prouve que ce type de représentation est bien approprié pour la construction de structures musicales hors contexte. Cependant, il n'est pas très adapté pour l'arrangement du matériau dans le temps.

- Inclusion de programmes dans les éditeurs :
Dans cette approche, les sous-structures de la partition peuvent éventuellement être le résultat d'un programme visuel. Cette deuxième modalité d'intégration est plus compréhensible à l'aide d'un exemple.

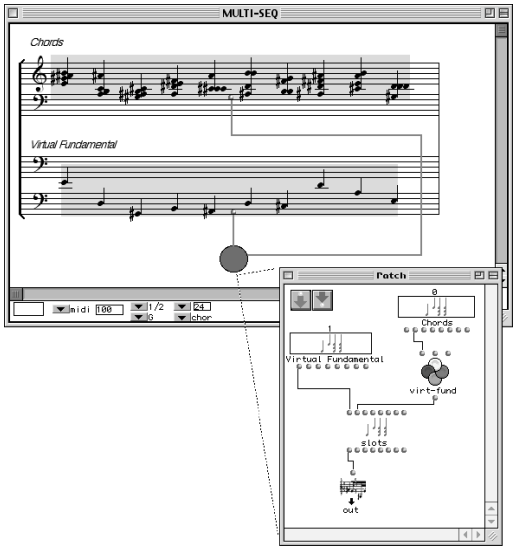


Figure 14. Programmes visuels inclus dans la partition

Dans la Figure 14, nous visualisons une polyphonie composée de deux séquences d'accords, *Chords* et *Virtual fundamental*. Ces deux séquences sont mises en relation par un programme visuel représenté par un cercle. A l'exécution de ce programme, la deuxième voix devient une mélodie où chaque note est la fondamentale virtuelle de l'accord qui lui correspond dans la première séquence. La partition, plus que le programme, nous semble l'espace de travail naturel pour un compositeur. Cette approche favorise la mise en relation des structures locales à la partition. Cependant, l'inclusion de programmes a généré des problèmes de visualisation et de navigation dans les éditeurs.

- La maquette :

La maquette est un concept nouveau visant à unifier les deux modalités précédentes. Le compositeur peut considérer la maquette comme une partition ou comme un programme. A la base, la maquette est une surface en deux dimensions contenant différents blocs que nous appellerons boîtes temporelles. La Figure 15 montre une maquette réalisée par le compositeur Mikhaïl Malt.

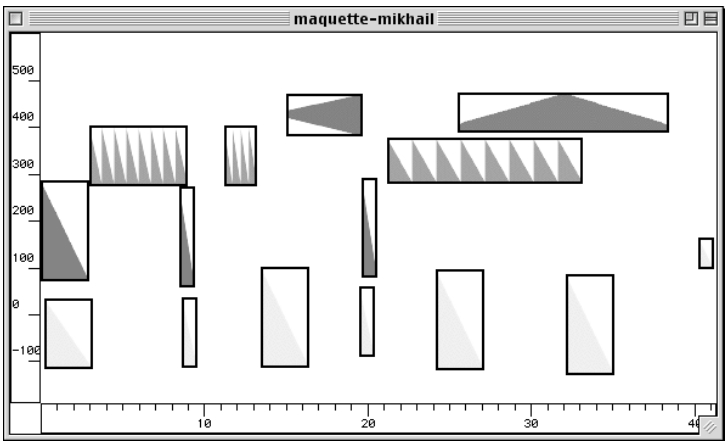


Figure 15. Exemple de maquette

La position horizontale des boîtes correspond à des dates en temps absolu. Leur extension horizontale correspond à leur durée. Dans cette exemple leur extension verticale figure l'intensité.

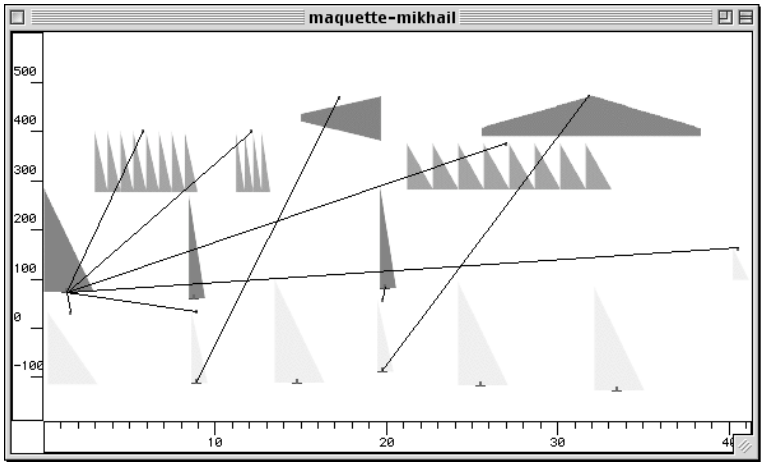


Figure 16. Connexion entre les boîtes temporelles

Dans la Figure 16, les connexions entre boîtes temporelles ont été révélées. Elles sont porteuses d'un autre niveau de sémantique musicale. On peut en effet y voir que les blocs se déduisent les uns des autres par des relations fonctionnelles. Chaque boîte a associé un programme visuel qui lui construit une structure musicale. On accède à un troisième niveau de l'organisation musicale, qu'on pourrait qualifier de syntaxique. Finalement, dans la Figure 17, les valeurs calculées pour chaque boîte peuvent être visualisées en utilisant la notation traditionnelle.

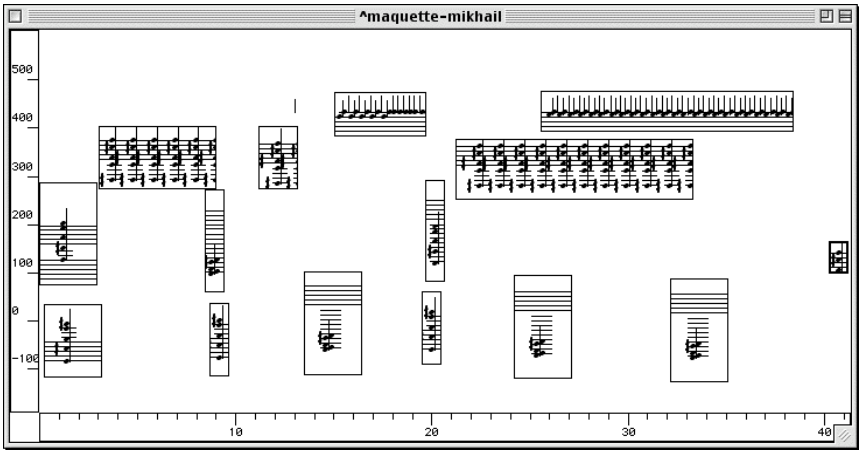


Figure 17. Notation musicale dans la maquette

Le lecteur pourra se référer à *[9]* pour une description approfondie des maquettes. D'autres travaux autour de la notation peuvent également être trouvés dans *[10]*. De nombreuses voies restent à explorer dans ce domaine ; la partition potentielle telle qu'elle est définie par Gérard Assayag dans [Ass93] est encore inachevée.

1.6 L'écriture du son : un problème nouveau

Il existe un grand nombre de compositeurs qui, sans aucune maîtrise technique, veulent se lancer dans la synthèse du son. Quand on parle de synthèse, on est obligé de parler d'ordinateurs. Celle-ci est sans doute l'un des apports les plus révolutionnaires de l'informatique en musique. La production de sons qui ne sont pas le modèle d'une cause physique a remis en cause le matériau sonore de la musique. Le compositeur néophyte en matière d'informatique musicale se trouve donc devant deux problèmes : le premier est la méconnaissance des modèles de synthèse pour produire le son ; le deuxième est l'incapacité à appliquer ses connaissances d'écriture symbolique à la synthèse. Le premier problème nous semble incontournable, on ne peut que recommander la lecture des articles sur la synthèse du son et de « jouer » avec quelques applications. Quant au deuxième inconvénient, même s'il semble moins important, il a été une source importante de mes recherches ces cinq dernières années.

Mes travaux dans ce domaine sont centralisés principalement autour du projet OMChroma *[21]* en collaboration avec le compositeur Marco Stroppa. L'un des principaux apports que l'informatique offre au compositeur est la réunion sous un seul paradigme des univers macro et micro compositionnels. Dans cette optique, le son est pensé comme le résultat d'un processus logique. Cependant, de nos jours, les logiciels d'aide à la composition sont plutôt consacrés à l'un ou l'autre de ces deux univers. En effet, la division entre CAO et traitement sonore reste une question d'actualité. En général, les travaux mettant en relation CAO et traitement du signal se contentent de générer des paramètres de synthèse, tout en gardant bien définie la frontière entre les deux domaines (on parle alors de contrôle de la synthèse [Tau02]). Il existe une autre famille d'environnements, [RoC85], [EcG94], qui a pour but l'intégration sous un même programme des aspects symboliques et d'autres liés à la synthèse. L'axe de mes recherches se dirige donc vers la réduction de la dichotomie entre un son vu comme le résultat d'un processus de synthèse ou comme une entité logique musicalement pertinente. Mon principal objectif est la définition et la mise à disposition de diverses abstractions et leur mise en relation avec des structures musicales de haut niveau (i.e. rythmes, accords, polyphonies, etc). Ces abstractions doivent surtout servir d'interface pour la génération des divers paramètres de synthèse.

La conception d'un tel outil de composition implique une réflexion sur un certain nombre de difficultés. Tout d'abord, la plupart des méthodes de synthèse numérique nécessitent la gestion de très nombreux paramètres. D'où le besoin d'outils automatiques de génération de ces données, qu'il serait trop fastidieux de spécifier « à la main ». Une autre difficulté inhérente à l'utilisation musicale des synthétiseurs est la difficulté d'obtenir un résultat « vivant », auditivement aussi intéressant que le son produit par un instrument de musique ou une voix humaine. On cherche à éviter des sons qui révèlent trop rapidement leur nature synthétique, et pour cela il faut utiliser soit des synthétiseurs très élaborés, soit un contrôle très fin des paramètres. Le matériau premier de la musique étant le temps, nous devons disposer de fonctionnalités permettant de le manipuler. C'est ce qu'attend un compositeur qui par sa pratique est habitué à ce travail sur le temps, et à le penser selon différents modes. Finalement, pour les pièces mixtes (synthèse et instruments), le compositeur est confronté aux rapports entre les paramètres de synthèse et son écriture. Un exemple typique est l'interaction entre les enveloppes d'amplitude et les dynamiques de la partition. Un changement dans l'un de ces paramètres a forcément des répercutions sur l'autre. Cette interaction nécessite des systèmes de spécification et de contrôle inédits, elle représente un enjeu majeur dans notre recherche. *OMChroma* sera repris plus en détail dans la section 3.2

1.7 Un problème essentiel : le compositeur

Tout au long de mon activité de recherche, j'ai entretenu des relations directes avec des compositeurs de divers horizons dans et hors les murs de l'Ircam. En tant que non-musicien, ces relations me semblent indispensables au maintien d'un principe de réalité musicale. La manière dont chaque compositeur conçoit les rapports entre informatique et musique est unique et cela à différents niveaux. Premièrement, par la différence des connaissances informatiques : il y a des compositeurs pour lesquels l'ordinateur est un chapeau magique duquel ils peuvent sortir à peu près tout. À l'opposé, se trouve le compositeur qui distingue ce qu'il est raisonnable de formaliser et d'implémenter, et ce qui relève de la science-fiction. En deuxième lieu, concernant la musique, il existe toute une gamme de compositeurs situés entre ceux qui ne discutent jamais des questions musicales, et ceux qui ont intégré l'ordinateur dans leur esthétique et dans leur stratégie créatrice. Finalement, la très grande diversité des modèles esthétiques qui coexistent chez les compositeurs aujourd'hui impliquerait la construction d'un système informatique par compositeur. Pour cela, je pense qu'il est nécessaire d'intégrer le compositeur comme un problème de plus (il s'agit d'une métaphore) dans mon projet de CAO.

Nous avons vu dans ce chapitre que la collaboration entre informaticiens et compositeurs va plus loin que l'imposition d'outils par l'informaticien ou l'exécution mécanique d'une commande du compositeur. Concevoir le compositeur comme un *end-user* est un non-sens, car la facilité et l'économie de temps ne sont pas ses préoccupations principales, mais aussi parce que le compositeur a du mal à déléguer les responsabilités de son acte de création, qu'il veut à tout prix garder individuel. Une telle approche placerait au centre de nos collaborations l'aspect technique et non celle de la création. Concernant la programmation, les utilisations actuelles de l'informatique dans la musique ont besoin de la présence d'un expert là où il n'y en aurait pas besoin.

Certaines pratiques informatiques ne sont pas déterminantes dans la composition, elles restent au stade d'outils sans influence dans l'esthétique du compositeur. Pour d'autres, on peut établir des comparaisons : en symétrie avec ceux qui pensent que programmer est un acte artistique [Lev92], je pense qu'il y a de la programmation dans la composition. Ainsi, je propose la figure du compositeur-programmeur pour que l'ordinateur puisse avoir un rôle irréductible dans la composition. Dans une proposition aussi contraignante, deux remarques doivent êtres faites : en premier lieu, ceci n'est pas une condition nécessaire - il bien évident qu'on peut composer sans l'ordinateur. Deuxièmement, cette condition n'est pas suffisante. Proposer un langage de programmation aux compositeurs n'est qu'un premier pas vers un emploi « plus intelligent » de l'informatique dans la création musicale.

Nous proposons donc un langage de programmation aux compositeurs (OpenMusic), ainsi que la logistique pour les orienter vers la programmation. Basés sur de multiples cas d'utilisation d'OM, les chapitres qui suivent vont s'intéresser à l'analyse des points forts ainsi que des limitations de notre approche. Nous essayons par la suite de montrer que le compositeur n'est pas imperméable à l'influence de l'informatique. La musique contemporaine, telle qu'elle est pratiquée à l'Ircam, est aussi le résultat de diverses recherches scientifiques. Il s'agit donc de définir la place de l'informatique dans cette période musicale, et plus généralement dans les activités liées à la création artistique.

2. Langages de programmation et CAO

Dans la composition musicale, l'outil informatique prend en charge principalement la formalisation et la construction du matériau selon des procédures généralisables et reproductibles - ce qui ne préjuge pas de la liberté de sa mise en oeuvre dans la partition finale. Cela est principalement dû au fait que les calculs permettant de construire les systèmes spécifiés par les compositeurs (systèmes harmoniques, rythmiques, mélodiques ou de timbre) sont le plus souvent complexes. Les langages de programmation permettent de définir des modèles informatiques utilisables par des compositeurs désireux de préparer des matériaux complexes, structurés par un ensemble de règles exprimées de façon cohérente. Il est à noter qu'on pourrait s'occuper, avec l'aide du compositeur, de la construction de ces modèles, mais, comme on l'a déjà remarqué, il faudrait écrire un programme de CAO pour chaque compositeur et, dans le pire de cas, un pour chaque pièce. C'est là où les langages de programmations me semblent incontournables. Ils permettent de matérialiser sous la forme de programmes les diverses abstractions du compositeur.

2.1 La formalisation

La formalisation musicale est une phase préalable à la conception et à l'écriture d'une ou de plusieurs pièces. Elle servira de langage musical pour un ou plusieurs compositeurs. Nous pouvons discerner deux approches de la formalisation musicale. L'une se détermine entièrement à partir de problématiques purement musicales et cherche l'outil mathématique qui sera le mieux adapté, et l'autre opère par transfert de connaissances d'un domaine extramusical au domaine de la musique [Ass00]. Quant à la pertinence d'une formalisation, elle dépendra pour le premier cas de la problématique musicale à formaliser. Dans le deuxième cas, elle devra trouver des correspondances entre les entités extramusicales (e.g. d'ordre mathématique, social, religieux, d'actualité, etc.) et des constructions musicales relativement signifiantes. Mais une logique formelle n'implique pas forcément une logique musicale. Pour que des correspondances de ce type puissent être utilisées dans une composition, elles doivent être soutenues par une conceptualisation et une pensée musicale [Mal01].

Formalisation de problématiques purement musicales

Un exemple de ce type de formalisation informatique est celle de le *set-theory*. Dans *[4]*, nous montrons les aspect théoriques et l'implémentation dans OM de la *Pitch-Class Set Theory,* qui est considéré comme un cas particulier de l'action d'un groupe (dihédral) sur un ensemble (Z). Le compositeur Paul Nauert dans sa pièce *A Collection of Caprices* pour piano *[om25]* utilise des classes d'équivalence de hauteur pour créer un sens d'uniformité à l'intérieur d'une

section et de contrastes entre les différentes sections. Les classes d'équivalence d'accords sont formées par rapport à leur contenu intervallique.

Un autre exemple de formalisation est proposé dans *[om4]* par le compositeur Jacopo Baboni. Cette approche, que l'on peut qualifier de grammaticale, consiste à définir de manière hiérarchique sa vision d'une composition. A la base, il existe des sons, l'articulation de deux *sons* dans le temps produit des *sonemes*, les *sonemes* à leur tour sont contenus dans des *morphemes*, qui composeront des *stylemes*, et ainsi de suite, jusqu'à la construction d'une œuvre musicale. L'implémentation de cette formalisation dans OM permet à Jacopo Babonni d'avoir un mécanisme homogène pour le contrôle local et global de son œuvre. Diverses pièces ont été créées par le compositeur en s'appuyant sur cette formalisation.

Un dernier exemple exposé ici est la formalisation ensembliste de la notion de suites modales du compositeur Anatole Vieru *[2]*. Les suites modales sont des « listes » cycliques qui peuvent être interprétées, soit comme des hauteurs, soit comme des intervalles (Figure 18).

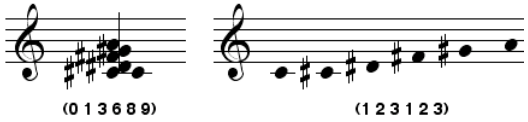


Figure 18. Interprétation d'une suite modale comme un accord et comme une gamme

Notre implémentation dans OM des suites modales de Vieru permet aussi une application dans le domaine rythmique (Figure 19).

Figure 19. Interprétation d'une suite modale comme un rythme

Finalement, nous avons donné dans *[15]* des algorithmes qui permettent de décomposer n'importe quelle suite modale en la somme d'un suite réductible (i.e. après *n* itérations la suite devient nulle.) et d'une suite reproductible (i.e. après *n* itérations la suite devient elle-même).

Formalisation de l'extramusical

Nombreux sont les exemples de ce type de formalisation. Alexander Mihailic, dans sa pièce *Crystals [om24]*, s'inspire des divers groupes définis dans la cristallographie : *triclinic*, *monoclinic*, *orthambic*, *trigonal*, *quadratic*, *hexogonal*. Dans son travail, il s'intéresse aux symétries pour les appliquer aux divers paramètres musicaux : hauteurs, dynamiques, spatialisation, et autres. Ces symétries produisent des espaces tridimensionnels où l'axe des *x* représente le temps et les deux autres sont pris parmi les paramètres déjà mentionnés. Le groupement de ces espaces constitue sa composition.

Dans les pièces *Ungrounded Zonnestraal* et *Zonnestraal [om15]*, le compositeur Christian Jaksjø s'impose de créer une musique qui reflète l'architecture. Plutôt qu'une formalisation des paramètres architecturaux, comme la matière, les densités ou le volume, il veut que sa musique rende compte des « interactions et transformations qui définissent l'architecture comme une unité ». Mis à part le côté artistique, ce qui me semble intéressant dans ce travail est le questionnement sur les *mapping* entre paramètres des divers domaines. Il est clair qu'un

paramètre architectural peut être facilement associé à un paramètre musical. On peut céder facilement, par coïncidence de langage, à l'association de notions comme la forme, le rythme ou le matériau. Mais, une simple correspondance des paramètres est-elle suffisante pour que la musique reflète l'organisation architecturale ?

Afin d'obtenir des gestes musicaux organiques, Johannes Kretz, dans sa pièce *Second Horizon* pour Piano et Orchestre *[om16]*, se base sur la simulation de mouvements physiques. A l'aide du logiciel MAX, il a formalisé le mouvement d'une balle, dans une salle close, soumise à des facteurs comme la pesanteur ou la friction. Ensuite, diverses trajectoires ont été importées dans OM pour servir de paramètres pour l'écriture. La traduction des positions spatiales dans le domaine harmonique n'a pas donné de résultats satisfaisants aux yeux du compositeur, mais les diverses enveloppes spatiales ont influencé la forme finale de l'œuvre.

Formalisation à partir des mathématiques

Les formalisations issues des mathématiques sont moins facilement classifiables dans les deux catégories précédentes. Répondre à la question de savoir si la musique est une discipline mathématique ou non ne relève pas de mes compétences ni de mon intérêt. Cependant, on ne peut pas nier l'influence du nombre comme source d'inspiration pour la composition. Le nombre d'or, la série de Fibonacci ou les carrés magiques en sont quelques exemples récurrents. Dans la pièce *Tombeau de Marin Mersenne* pour luth et synthétiseur *[om2]*, le compositeur Michel Amoric utilise OM pour calculer, à partir des nombres de Mersenne ($M= X^n - 1$), les analyses spectrales des instruments simples. L'exemple suivant montre une utilisation plus poussée des mathématiques comme « étincelle » pour la création.

Dans l'œuvre *Nomos Alpha* du compositeur Iannis Xenakis, l'ordre temporel des objets sonores de la pièce est géré par une structure algébrique particulière : le groupe des 24 rotations du cube dans l'espace. Ces objets, au nombre de huit, sont mis en correspondance avec les huit sommets du cube. Cette correspondance est rendue possible en prenant un cube de référence qui donne un ordre privilégié aux huit sommets du cube. Toute rotation du cube induit une permutation des sommets du cube unitaire, donc un ordre particulier dans l'enchaînement des huit complexes sonores. La propriété de fermeture et le fait que le groupe des rotations du cube dans l'espace est fini, permettent de construire des séquences cycliques de rotations selon le processus de Fibonacci. Autrement dit, chaque élément de la séquence (c'est-à-dire chaque rotation) est le produit des deux précédents. La formalisation du processus de Fibonacci, que Xenakis utilise dans le groupe des rotations du cube, a révélé des propriétés qui ne sont pas évidentes à démontrer d'un point de vue mathématique et qui ont des retombées musicologiques très significatives [And03]. Les chaînes engendrées par le processus de Fibonacci appliqué aux 24 rotations du cube ont toujours un comportement cyclique duquel il n'est pas évident de donner a priori la période (longueur) ni le nombre d'éléments différents. Quelques résultats ont été obtenus à partir d'une exploration de l'implémentation dans OM : dans un espace de 576 possibilités, les boucles de Fibonacci ont une longueur maximale égale à 18. Parmi ces boucles, 5 est le nombre minimum d'éléments répétés dans une boucle. L'ensemble de ces boucles contient 216 éléments.

Xenakis a choisi ses boucles de Fibonacci à l'intérieur de cet ensemble et c'est précisément ce choix qui détermine la forme globale de la pièce, divisée en 18 parties, chaque partie correspondant à un élément du groupe des rotations. La suite de Fibonacci utilisée par le compositeur, de longueur et degré maximaux, est représentée dans la Figure 20.

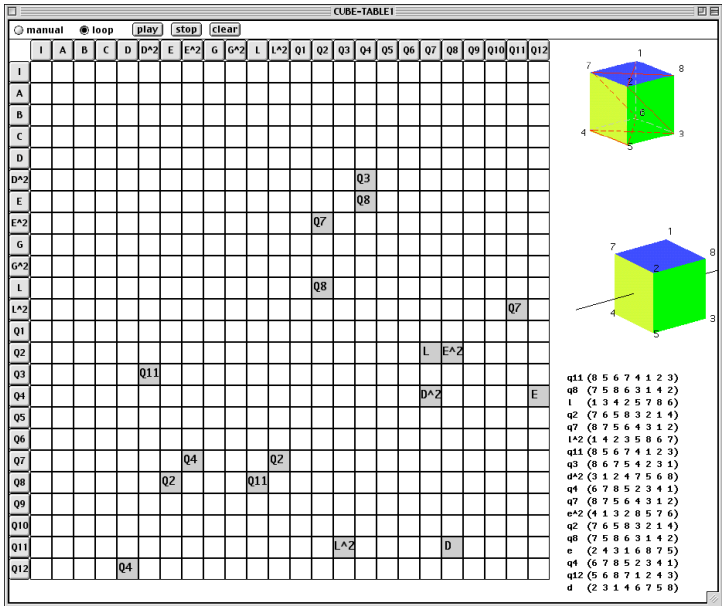


Figure 20. Permutations des huit sommets d'un cube, utilisées pour Nomos Alpha

2.2 La modélisation

Au centre de notre approche de la CAO se trouve la notion de modèle informatique. De manière générale, le modèle est un dispositif formel qui, rendant compte, au moins partiellement, des caractéristiques d'un processus matériel, en autorise expérimentalement la simulation aux fins de vérification, d'observation ou encore de production de processus similaires. Mais, comme il est dit dans [Cou93] il ne faut pas croire que parce qu'il existe un algorithme pour le calcul d'un objet musical, le compositeur va l'utiliser systématiquement sans se poser de questions. En réalité, les modèles simplifient le problème musical, ce qui implique des choix arbitraires. Nous avons souvent remarqué qu'à partir de modèles existants, la tendance à modifier aussi le modèle lui-même est grande. L'emploi que nous faisons ici du mot modèle est alors à mi-chemin entre le sens scientifique et le sens artistique du terme.

A titre d'exemple, nous présentons la modélisation informatique de la pièce *Nomos Alpha*. Pour cette réalisation, nous sommes partis des éléments systématiques décrits par le compositeur tout en essayant d'explorer le champ des potentialités théoriques offertes par le système. L'implémentation dans OM permet de mettre en évidence le processus de manipulation *hors temps* d'outils algébriques et leur organisation *temporelle* de la part du compositeur. La maquette (programme + notation) permet d'étudier les différents enchaînements temporels d'objets musicaux dont les propriétés ont été décrites dans la section 2.1. Ce processus est visualisé à l'aide des emboîtements de maquettes à l'intérieur d'une maquette principale (Figure 21).

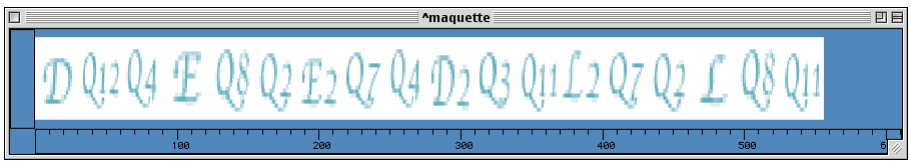


Figure 21. Suite des rotations dans Nomos Alpha

Chaque bloc, étiqueté par une lettre, est aussi une maquette contenant une permutation des huit sommets du cube. A chaque sommet, Xenakis associe un complexe sonore C_i. Les divers complexes sont décrits dans la Figure 22.

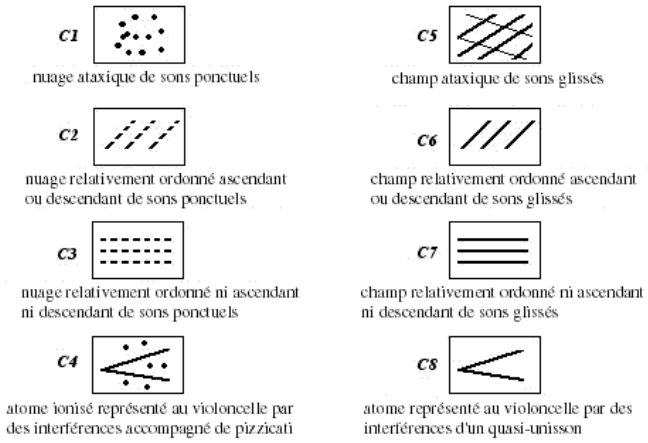


Figure 22 : Les huit complexes sonores de la pièce « Nomos Alpha »

La Figure 23 montre la séquence des huit complexes sonores attachés à l'opération de groupe indiqué par la lettre *D,* et qui avec *Q12* ont été choisies comme point de départ de la pièce.

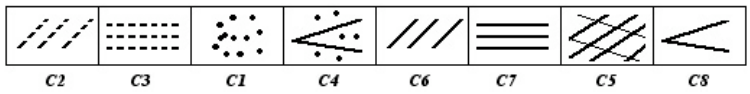


Figure 23. La séquence initiale des complexes sonores

La Figure 24 offre une nouvelle représentation de la première permutation (D) sous forme de maquette. Cette fois, les complexes sonores sont représentés par des carrés de dimensions différentes et ayant différents niveaux de gris. Plus un carré est clair, plus la valeur de densité du complexe sonore correspondant est grande. Intensités et durées sont représentées respectivement par la dimension verticale et la dimension horizontale. Par exemple, les deux premiers complexes sonores ont les mêmes valeurs d'intensité et de durée mais deux valeurs de densité différentes.

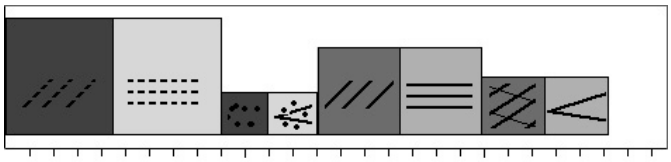


Figure 24. Représentation d'une permutation sous la forme de maquette

Notons que la modélisation informatique permet d'obtenir d'autres variantes de la pièce, simplement en utilisant des séries cycliques de Fibonacci différentes. C'est une démarche qui va pleinement dans le sens des stratégies compositionnelles que Xenakis a explicité dans ses écrits théoriques, lorsqu'il décrit une famille de suites ayant les mêmes caractéristiques (longueur et degré) que celle utilisée dans l'architecture formelle de la pièce. Les changements

d'autres paramètres, tels que la nature des complexes sonores ou le polyèdre de base, permet une vaste stimulation du modèle.

La modélisation sous forme de maquette permet l'accès à trois niveaux interdépendants d'organisation musicale dans Nomos Alpha:

- L'emboîtement des maquettes.
- La forme donnée par chaque permutation
- Le niveau du matériau de base.

Ces trois axes d'organisation sont autant de logiques musicales qui s'interpénètrent, notre modèle de maquette permet de matérialiser cette interpénétration et d'en donner un contrôle interactif au compositeur. Ce contrôle débouche sur des possibilités d'expérimentation combinatoire inédites : les blocs peuvent être réarrangés, dilatés ou compressés dans le temps ; ils peuvent s'emboîter de différentes manières ; la nature des complexes sonores peut être redéfinie, ils peuvent se définir par un programme visuel à l'intérieur des blocs.

2.3 L'expérimentation

Une fois le modèle implémenté, il est peu probable qu'un compositeur pense que sa pièce se limite au résultat d'un paramétrage donné. A la différence d'autres approches (la musique algorithmique, par exemple), dans notre conception de la CAO, le matériau résultant d'un calcul est retravaillé par le compositeur en fonction d'autres critères, esthétiques pour la plupart. A la différence d'une approche artisanale, dans l'art, le résultat final n'est pas connu à l'avance. Le matériau issu d'un modèle devient alors matériau potentiel, c'est au compositeur de mettre « fin » à la boucle de stimulation du modèle. Un modèle génère une famille de résultats pouvant être étendus progressivement, il peut être vu comme une classe d'équivalence de compositions. A l'opposé, deux résultats identiques ne provenant pas du même modèle ne sont pas égaux, ils n'ont pas les mêmes possibilités d'évolution. Si la partition dénote l'œuvre, le modèle lui donne sens.

La fonction principale d'un modèle compositionnel est son utilisation pour formuler des hypothèses et les stimuler. La facilité d'expérimentation devient alors essentielle dans notre démarche. Les systèmes de synthèse en temps réel sont un exemple réussi : en même temps que le compositeur modifie les paramètres de synthèse, il peut écouter le résultat de ses modifications. Des « nouveaux instruments » sont ainsi créés grâce à l'association des procédés de synthèse, des capteurs gestuels et de l'interactivité. Dans notre approche de la CAO, ce type d'interaction en temps réel semble conceptuellement impossible (voir section 5.4). Mais cela n'est en aucun cas une raison pour la négliger. Il est clair, qu'un système peu performant, avec une syntaxe complexe et où le compositeur doit gérer la mémoire ne faciliterait pas une bonne interaction. Voici quelques aspects qui doivent être pris en compte pour aider à l'expérimentation.

L'environnement de programmation

OM a été implémenté en Lisp. Les programmes graphiques sont traduits en code lisp et ensuite interprétés. Ainsi, l'environnement de programmation graphique bénéficie d'un court cycle de programme et permet la compilation « à la volé ». L'interface principale dans OM est

le *workspace*. Le *workspace* est un gestionnaire icônique de fichiers assurant la persistance de l'environnement (Figure 25).

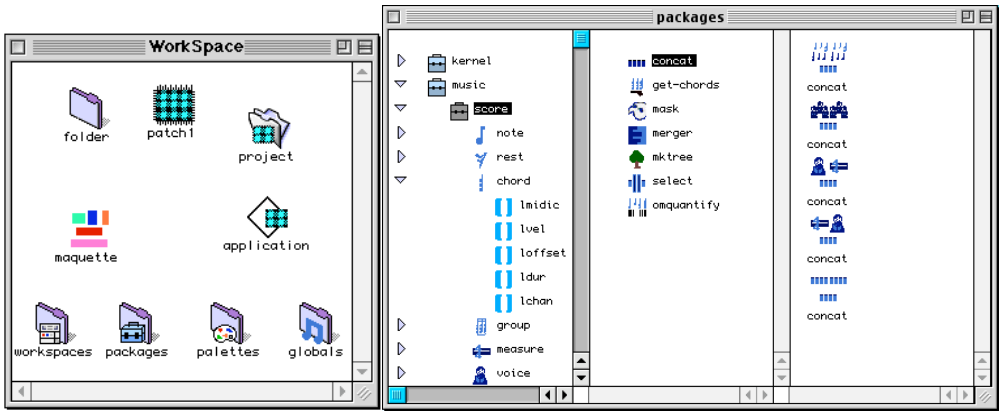


Figure 25. Interface OM

Un *workspace* contient principalement : des *folders* permettant d'organiser hiérarchiquement l'information ; des programmes visuels (*patches* ou *maquettes*) ainsi que des ensembles des classes et des fonctions génériques réunis dans des *packages*.

Le *workspace* est né de la nécessité fondamentale pour le compositeur de développer son travail dans différentes directions par rapport à la complexité et à la spécificité de ses constructions. Voici deux exemples : une approche *top-down* dans *[om25]* où le compositeur décide que sa pièce consistera en 11 sections qui alterneront sections courtes et longues, et il les écrit ensuite. Une approche *down-top* dans *[om31]* où le compositeur écrit des motif rythmiques pour après établir la forme de la pièce en concaténant les motifs.

Le *workspace* est aussi responsable de l'accès aux données musicales et de leur persistance. Cela peut être d'une grande importance pour certaines approches de la CAO. Pour son opéra « K » *[om23]*, Philippe Manoury fait appel à des techniques d'écriture post-sérielles. Ces contraintes ou règles d'écriture musicale peuvent être aisément automatisées, permettant la génération automatique de matériau musical. L'utilisation de OM, dans ce cas, sert à la construction des « réservoirs » de mélodies, d'accords et de rythmes dans lesquels le compositeur va puiser pendant l'écriture de la partition.

Visualisation

En addition aux aspects de notation exposés dans la section 1.5, la visualisation de données extramusicales peut s'avérer utile pour le compositeur. Un exemple intéressant de ce type de visualisation a été développé par Jean Bresson [Bre03]. Il s'agit d'un éditeur permettant la représentation et l'édition de descriptions sonores, exprimées dans le format standard Sdif [SwW00]. Une caractéristique de ces descriptions sonores, généralement issues de l'analyse, est la grande quantité de données qu'elles peuvent contenir. Dès lors, une partie du travail du compositeur consiste à réduire ces données (par filtrage, sélection,…) afin d'obtenir des données à la fois représentatives du son et utilisables dans un contexte musical symbolique. En effet, pour le compositeur, l'évolution d'un paramètre (e.g. une enveloppe d'amplitude) peut être plus intéressante que les valeurs précises de ce paramètre. La Figure 26 montre un exemple d'une représentation spectrale tridimensionnelle proposée par l'éditeur Sdif.

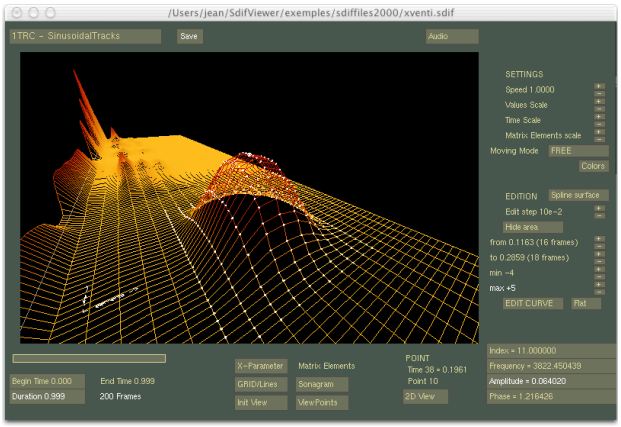


Figure 26. L'éditeur Sdif

Trois possibilités de manipulation sont offertes par cet éditeur :

- graphique, sur l'image tridimensionnelle, par mouvements, transformations, projection, etc.
- par programmation, en utilisant des algorithmes graphiques dans OM
- auditive, en associant des informations sonores aux données représentées.

Le contrôle graphique de ces données permet alors des appréciations subjectives et peut influencer les choix esthétiques du compositeur.

2.4 Diversité des applications

Grâce à leur généralité, les langages de programmation sont applicables aux diverses phases de la « production » des œuvres, en allant des plus conceptuelles aux plus matérielles. On a vu jusqu'ici des applications à la formalisation et à la conception de l'œuvre, nous allons voir maintenant comment ils peuvent aussi intervenir dans l'écriture, la diffusion ou l'analyse musicale.

Traitement symbolique du son

Le travail de stage du DEA de Peter Hanappe [Han95] a consisté à créer une passerelle entre le domaine du son et le domaine symbolique. L'idée était d'analyser des sons pour en extraire une représentation symbolique qui sera exportée vers OM. Ce problème est connexe à celui de la transcription automatique, cependant le but n'était pas la reconstitution d'une partition existante. En effet, nous nous sommes situés dans un cadre de composition dans lequel on suppose que le musicien, partant d'un signal, désire s'en servir comme matériau générateur. Il va, à l'aide d'un ensemble de procédures combinant automatisme et interaction, en extraire des informations musicales qui lui permettront de constituer des structures mélodiques, harmoniques, rythmiques et dynamiques, représentables dans le cadre de la notation instrumentale. Ce type de démarche est largement utilisé ; voir *[om26]*, *[om21]*, *[om39]*. A titre d'exemple nous exposons le travail réalisé par Elaine Thomazi-Freitas dans sa pièce *Derrière la Pensée [om38]*.

Le matériau sonore de départ est composé de sons courts (de la mer et du vent) présentant une structure spectrale riche. Ces sons ont été analysés en utilisant le logiciel *Audiosculpt*, puis ces analyses ont été exportées vers OM. L'analyse additive a été représentée sous la forme de

séquences d'accords. Un filtre a été appliqué à cette séquence pour définir le matériau de composition. A l'aide des algorithmes (strech, filtres, rotations), la sélection et la manipulation harmonique ont permis de construire le matériau de base. Le *patch* de la Figure 27 illustre ce processus.

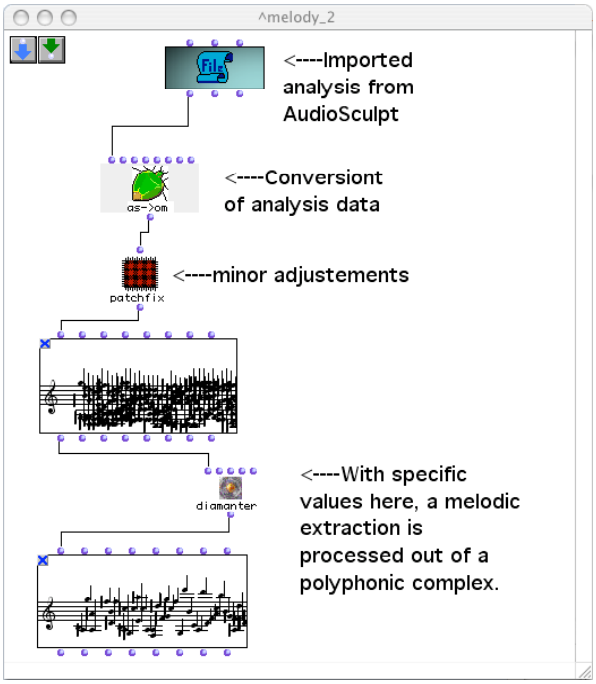


Figure 27. Construction du matériau dans *Derrière la Pensée*

La spatialisation

La spatialisation du son se présente comme un élément de base dans plusieurs compositions contemporaines. Bien qu'elle puisse être réalisée sans moyens électroniques, l'utilisation de l'ordinateur offre des possibilités inédites. C'est pourquoi nous considérons la spatialisation comme un autre paramètre compositionnel, apportant une dimension formelle supplémentaire que le compositeur peut manipuler.

L'implémentation de la bibliothèque *OMSpat [13]* utilise comme structure de base la matrice. Dans une matrice, les colonnes correspondent aux vecteurs de paramètres pour la spatialisation et les lignes aux sons à spatialiser. Dans la Figure 28, nous voyons la création d'une matrice à 3 colonnes (3 sons à spatialiser). Les trajets, courbes en trois dimensions, représentent un ensemble de points dans un espace tridimensionnel correspondant au parcours des sons. Les trajets peuvent être édités à la main, mais aussi à l'aide d'algorithmes graphiques.

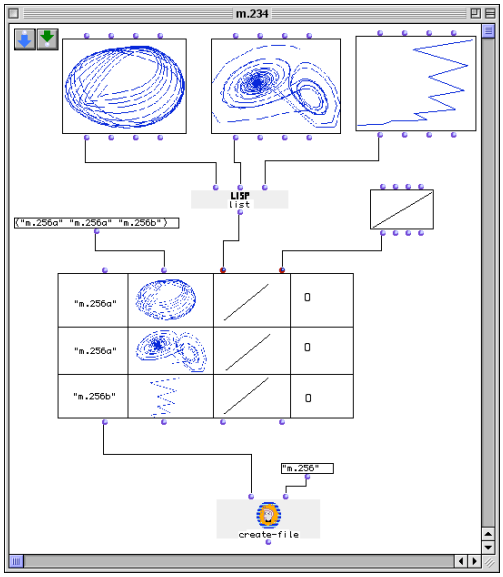


Figure 28. Spatialisation de trois sons

OMSpat a servi à la spatialisation de la pièce *Stelae for the failed times* du compositeur Brian Ferneyhough *[om27]*. Nous montrons ci-dessous (Figure 29) un exemple de la partition d'évènements sonores destinés à être spatialisés, et leur description spatiale.

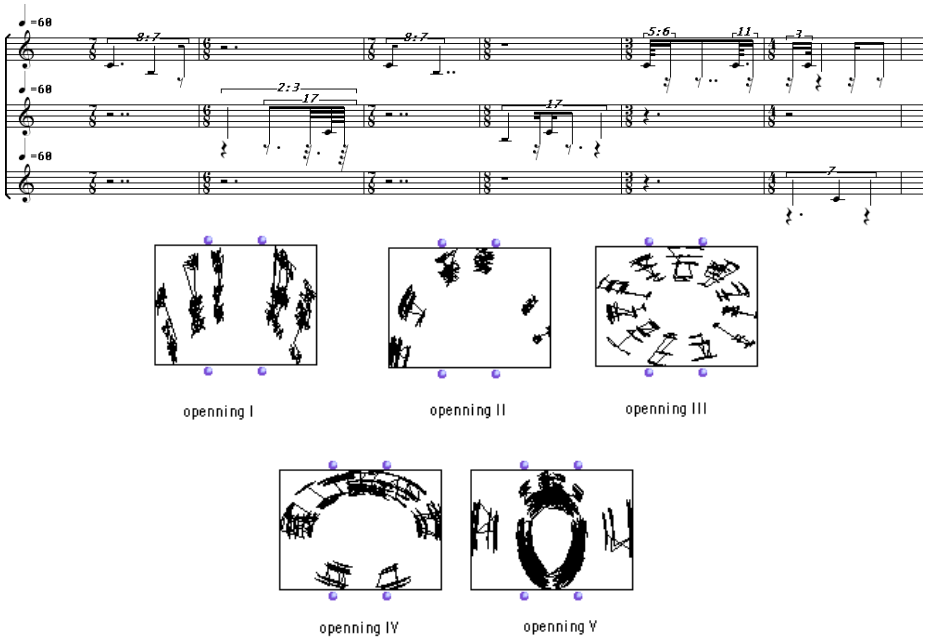


Figure 29. Trajets de spatialisation à partir de données musicales.

Dans l'exemple ci-dessus, chaque note est associée à un fichier son : les trajectoires sont organisées temporellement à partir de la partition. Les trajets ont été définis principalement par rapport à plusieurs paramètres musicaux (spectre, densité harmonique, durée, etc.).

Nous avons vu comment la CAO est appliquée aux divers stades de la vie d'une œuvre, conception, écriture et diffusion. Nous finissons ce chapitre en notant l'importance des langages de programmation pour relier ces diverses étapes durant la composition. Le compositeur peut ainsi se faire une idée du résultat final et vérifier ses intuitions.

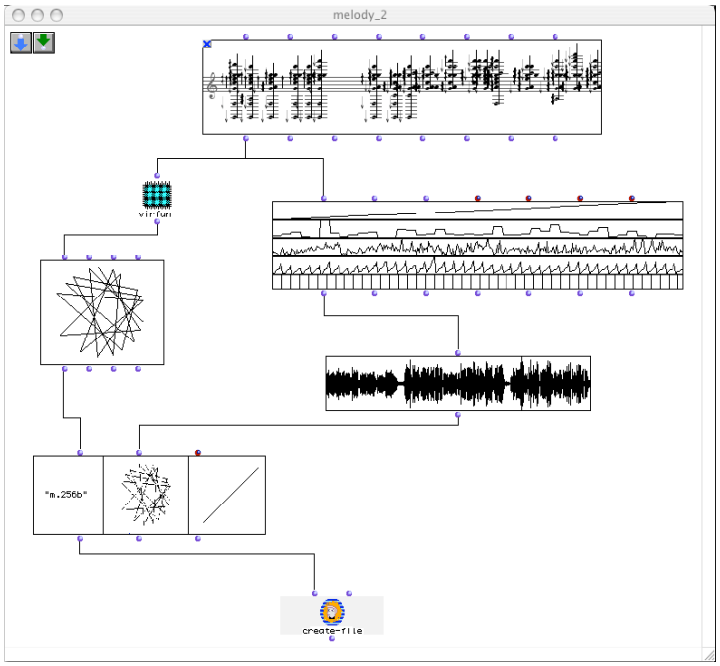


Figure 30. Ecriture, synthèse et spatialisation

L'exemple de la Figure 30 montre un son construit à partir d'une séquence d'accords. La spatialisation du son est en rapport avec le contenu harmonique de la séquence. Le changement d'une note aura pour conséquence le changement de la synthèse et de sa spatialisation.

3. Styles de programmation et CAO

Comme on l'a signalé dans l'introduction, l'utilisation d'un style de programmation déterminé plutôt qu'un autre suscite des expressions différentes. Le choix d'un style de programmation spécifique peut dépendre de la nature de l'application. Il semble qu'un seul paradigme de programmation ne convienne pas à tous les cas. Donc, il ne faut pas chercher à adapter à tout prix le paradigme au problème, mais au contraire choisir le ou les approches les plus appropriées selon la situation musicale. Il est donc primordial d'étudier plusieurs types de programmation. Dans ce chapitre, nous étudierons, à travers des exemples, trois styles de programmation : fonctionnelle, par objets et par contraintes. Nous essayerons d'analyser, pour chaque style, ses principaux atouts ainsi que ses faiblesses. Finalement, ce chapitre abordera le problème de l'intégration des divers paradigmes dans un seul environnement. Nous montrerons comment la programmation visuelle est proposée pour permettre une telle combinaison.

3.1 Programmation fonctionnelle

Il existe de nombreux projets fondés sur l'application du paradigme fonctionnel à la musique. Parmi les plus connus, citons les travaux de Mira Balaban où le lambda calcul est utilisé pour définir des structures temporelles hiérarchiques [Bal96]. Dans le domaine de la CAO, Elody [OrF97] est peut être le logiciel le plus représentatif de cette approche. Elody est un environnement de composition musicale basé sur un langage fonctionnel visuel qui permet la manipulation algorithmique et la transformation de structures musicales. Le principe général d'utilisation d'Elody consiste à construire des objets musicaux (lambda fonctions) en combinant des objets simples (notes ou silences).

Matériaux et processus compositionnels

Parmi les trois approches que nous étudierons dans ce chapitre, l'approche fonctionnelle est certainement la plus utilisée par les compositeurs. En effet, ce style de programmation donne une réponse à la nécessité de disposer d'objets multidimensionnels et de leur appliquer des opérations qui les modifient ou produisent de nouveaux objets. OM est, à l'origine, un langage fonctionnel et, dans ce sens, peut se concevoir comme une bibliothèque de fonctions implémentant des opérations musicales. Pour le compositeur, la classification des dites fonctions par rapport au type de connaissance musical qu'elles représentent est d'une très grande importance. Nous trouvons dans OM des fonctions dédiées aux intervalles ; des fonctions de traitement harmonique ; d'interpolation d'accords, de motifs et d'objets en général. Si la composition est considérée comme une interaction entre matériau musical et les processus qui le modifient alors le style fonctionnel se révèle pertinent par rapport à la pensée musicale. En effet, nous pourrons considérer le matériau comme des arguments, tandis que les processus de composition correspondront aux fonctions. Dans une approche strictement fonctionnelle, comme celle d'Elody, le matériau aurait pour représentation une fonction. Dans OM, des ty-

pes abstraits de haut niveau ont été définis afin de rendre le matériau plus ductile. Voici deux exemples d'application du paradigme fonctionnel pour la composition.

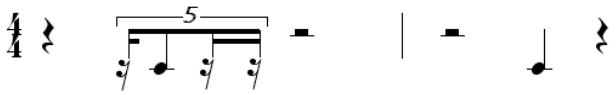
Structures rythmiques.
Pour des paramètres musicaux comme la hauteur ou les intensités, les structures des données ainsi que leurs fonctions peuvent se définir d'une manière simple. Très souvent, les entiers et quelques opérations arithmétiques sont suffisants. Pour les matériaux rythmiques, le problème s'avère plus compliqué. Nous avons défini une structure de données appelée arbre rythmique (AR). Un AR est défini comme un couple (**D S**) où **D** est une fraction (> 0) et **S** est une liste de n éléments définissant des proportions de **D**. Chaque élément de **S** peut être soit un entier, soit un **AR**. Une description détaillée de ces arbres rythmiques peut se trouver dans *[10]*. La figure suivante montre un arbre rythmique et sa représentation traditionnelle.

AR1 = (2 ((4/4 (1 (1 (1 -2 1 1)) 1 1)) (4/4 (1 (1 (1 2 1 1)) -1 1)))))

Diverses opérations (fonctions) sont ensuite définies pour les AR dont voici quelques exemples :

-Inversion : Toutes les notes sont transformées en silences et vice versa.

Inversion (AR1) = AR2 = (2 ((4/4 (-1 (1 (1 -2 1 1)) 2)) (4/4 (2 1 -1))))



-Rotation : Applique une rotation de n places aux attaques du rythme.

Rotation (AR1) = AR3 = (2 ((4/4 ((1 (1 (1 (-2 1 1 1)) 1 1)))) (4/4 ((1 (1 (1 (2 1 1 -1)) 1 1))))))

Sur cette représentation rythmique, les compositeurs Karim Haddad et Mikhaïl Malt ont construit une bibliothèque dans OM qui définit un ensemble pertinent d'opérations. Bien évidemment, à l'aide d'un programme visuel dans OM, tout compositeur peut définir ses propres opérations. La Figure 31 illustre un tel programme, consistant en l'inclusion d'un rythme dans un autre à une position donnée.

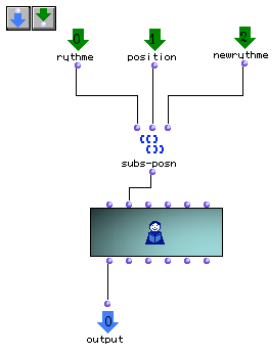


Figure 31. Inclusion d'un arbre rythmique dans un autre

Ainsi, en appliquant ce programme à AR1 et AR4 =  , nous avons :

Inclusion (AR1, 0, AR4) = AR5 = (2 ((4/4 ((1 (1 2)) (1 (1 -2 1 1)) 1 1)) (4/4 (1 (1 (1 2 1 1)) -1 1))))

Des exemples d'utilisation des arbres rythmiques pour la composition se trouvent dans *[om22]*, *[om27]*, *[om13]*.

Application à la synthèse sonore.

Dans OM, la démarche fonctionnelle a rapidement été adoptée par des compositeurs pour des problèmes symboliques, suite à quoi nous avons été tentés d'appliquer ce paradigme au domaine de la synthèse. C'est ce qui a été fait dans « L'Amour de loin », le premier opéra de la compositrice Kaija Saariaho [BaN03]. La Figure 32 résume le processus de transformation que subissent les matériaux compositionnels dans cette pièce.

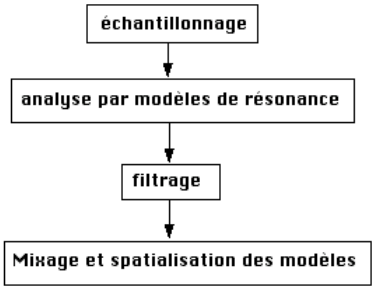


Figure 32. Transformation des matériaux dans L'Amour de loin

-Le matériau sonore est composé par des échantillons de voix parlées extraites du livret, et par des sons instrumentaux.

-Ces échantillons sont analysés en utilisant la technique de modèles de résonance. Le résultat de cette analyse est une liste de fréquences, amplitudes et largeurs de bande qui représente beaucoup d'information sur le timbre des sons (pour plus d'information sur les modèles de résonances, voir [PBJ86]).

-L'étape de filtrage consiste en un croisement entre une partie électronique issue d'une structure harmonique prédéfinie, et un timbre issu de l'analyse par modèles de résonances. La partie électronique est d'abord écrite sous forme de partition. Les sons électroniques reposent sur des structures harmoniques qui sont propres à chaque personnage de l'opéra. La Figure 33 présente les accords utilisés dans l'élaboration des sons électroniques pour le personnage Jaufré.

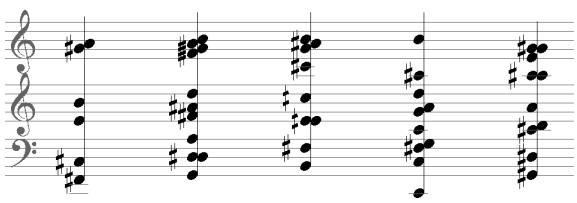


Figure 33. Accords de Jaufré

Le patch de la Figure 34 illustre l'extraction du modèle de résonance pour un échantillon d'une note de piano.

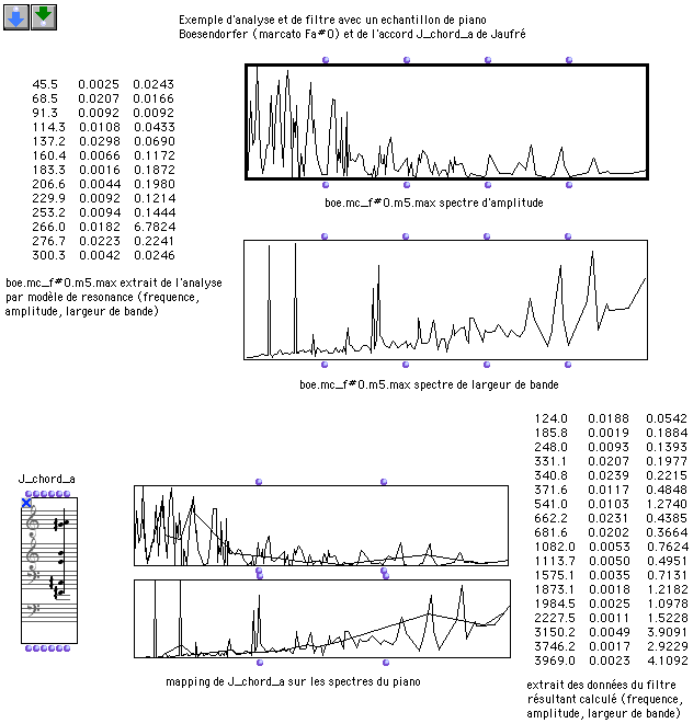


Figure 34. Analyse par modèles de résonances d'un son de piano

Dans la Figure 35, le premier accord du personnage Jaufré est utilisé. A chacune des fréquences des notes de l'accord sont associées, après un calcul d'interpolation spectrale, l'amplitude et la largeur de bande correspondantes du modèle instrumental. Le patch représenté sur cette figure effectue le filtrage.

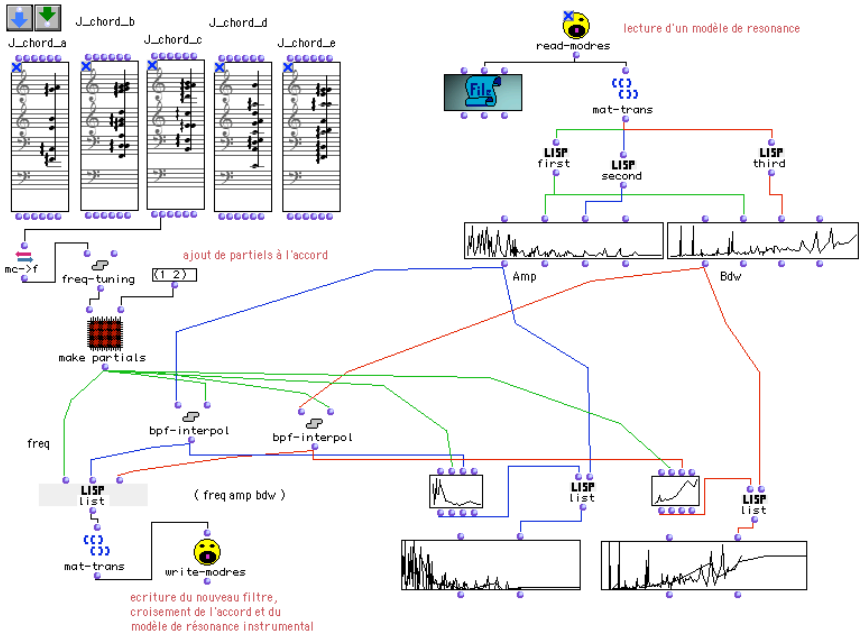


Figure 35. Patch pour le filtrage

Pour chacun des accords associés aux personnages, de nombreux filtrages ont ainsi été réalisés. Dans toute l'œuvre, environ trois cents filtres ont été générés. En considérant le nombre d'accords et d'échantillons instrumentaux analysés, il est aisé de voir que le nombre de combinaisons de choix possibles est très élevé.

-Finalement, le mixage des sons réalisés précédemment est effectué en temps réel à l'aide du spatialisateur développé à l'Ircam.

L'abstraction fonctionnelle.
Avant de conclure cette section, soulignons le rôle de l'abstraction fonctionnelle dans la composition. Pour des applications musicales, où l'activité du compositeur ne peut pas être connue à l'avance, l'abstraction est le mécanisme qui sert de fondement à la création. En effet, dans OM les patches sont conçus comme un espace de travail dans lequel le compositeur expérimente avec le matériau. Au fur et à mesure de cette interaction, certains éléments du programme se rendent variables [Mal02], pour arriver à la définition d'objets fonctionnels qui pourront à leur tour participer à d'autres abstractions. C'est en cela que réside, à notre sens, la pertinence de l'approche fonctionnelle en musique.

3.2 Programmation par Objets

Plusieurs applications en informatique musicale ont été développées sur les langages objets, parmi les plus connues nous pouvons citer [Pac94], [Pop91]. Elles sont pour la plupart des bibliothèques qui réifient des structures musicales et définissent un protocole pour leur manipulation. Cependant, les applications musicales qui fournissent une interface de programmation par objets adaptée aux compositeurs sont rares. En pensant que les notions de classe, polymorphisme, héritage ou réutilisation, pouvaient trouver un sens dans une démarche de composition, nous avons implémenté une interface graphique pour CLOS [Ste98]. Cependant, les résultats d'utilisation n'ont pas été à la hauteur de nos espérances. Peu de compositeurs se sont appropriés ces outils. Des notions comme l'héritage, qui en principe semblent « naturelles », ont été difficilement compréhensibles pour le compositeur, sans parler d'autres concepts tels que l'instanciation ou l'invocation des méthodes. Nous avons alors changé de stratégie et nous nous sommes concentrés sur l'utilisation de la programmation par objets pour la synthèse du son. L'utilisation des techniques par objets dans le système *OMChroma* s'est alors imposée naturellement. Dans *CSound* par exemple, les instruments peuvent être vus comme des classes et les *scores* comme des instances. Il existe diverses compilations d'instruments *CSound* [Bou99], mais en général, quand un compositeur veut faire une synthèse, il doit construire un instrument (définition d'une classe) ou modifier un instrument existant (héritage). Nous montrons ci-dessous les principes de base dans *OMChroma*.

L'idée principale dans *OMChroma* consiste à définir des entités de haut niveau pour le processus de synthèse de son. A la base de ces entités de synthèse, nous avons une représentation matricielle, où lignes et colonnes de la matrice sont associées aux paramètres de synthèse. Nous avons implémenté une classe abstraite appelée *array* avec un champ particulier (*numcol*) spécifiant le nombre de colonnes. La classe *array* n'est pas utilisable directement, elle doit être sous-classée en ajoutant des *slots* qui correspondront aux paramètres de synthèse et qui ajouteront des lignes à la matrice. Dans la Figure 36, nous voyons la création d'une matrice à 20 colonnes et 5 lignes (*numcol* = 20 et 5 *slots* ajoutés). Si les valeurs des paramètres

sont des nombres, les lignes pourront être visualisées sous forme de courbes. Une enveloppe sera clairement une matrice avec une seule ligne.

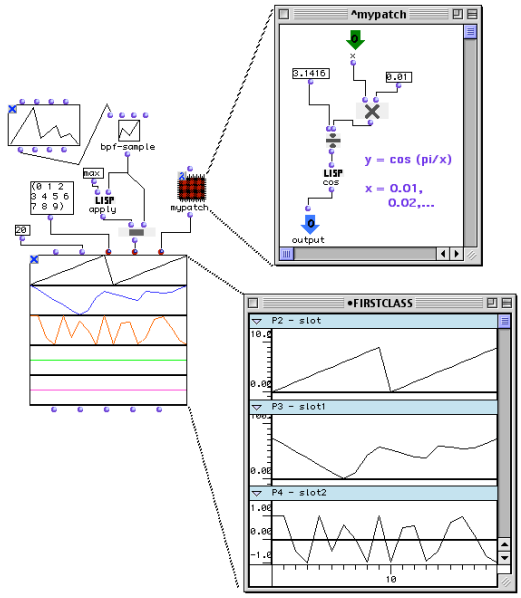


Figure 36. Une instance de la classe array

Il n'est pas nécessaire de remplir toutes les lignes. Dans notre exemple, seulement 3 des 5 *slots* ont été fournis. Pour les deux autres, on utilise les valeurs par défaut données lors de la définition de la classe, les courbes en question seront des constantes. Une particularité de la classe *array* est que les valeurs des *slots* peuvent être données sous diverses formes. Dans la Figure 36, les trois entrées de paramètres sont (de gauche à droite) : une liste d'entiers, répétée cycliquement jusqu'à atteindre le nombre de colonnes ; le résultat d'un algorithme visuel (patch) ou une lambda expression visuelle (montrée dans la fenêtre *mypatch*, y = cos(Pi/x)). Quand les valeurs des *slots* sont des fonctions, la matrice ne garde pas les valeurs de l'application de la fonction, mais la fonction elle-même. Les valeurs, produites à partir de l'application de cette fonction, seront calculées lorsque cela sera nécessaire (évaluation paresseuse).

Les *Arrays* peuvent être construits à partir d'objets musicaux. Pour cela il faut définir un algorithme d'interprétation, tel qu'il est montré dans la Figure 37.

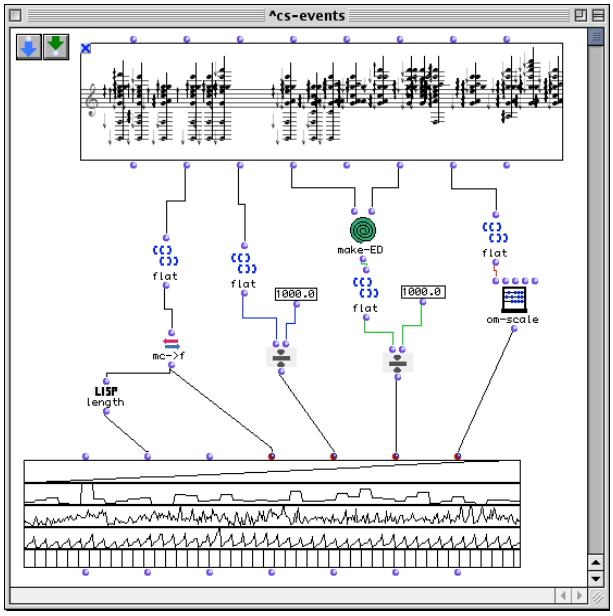


Figure 37. Création d'un *array* à partir d'un objet musical

Tout modèle de génération sonore a besoin de données de contrôle, qui dépendent du synthétiseur utilisé. Cependant, plusieurs paramètres de contrôle peuvent être conceptuellement les mêmes, indépendamment de leur implémentation. Un vibrato, par exemple, sera toujours utilisé comme un vibrato quelque soit la façon dont il est codé. Les *arrays* peuvent être groupés en sous-classes abstraites qui, grâce au polymorphisme, peuvent être interprétées par un synthétiseur donné. Ainsi, des données similaires n'auront pas besoin de structures différentes pour être traitées par différents synthétiseurs. Les algorithmes de production de données sont ainsi isolés du moteur de synthèse. Cette étape d'interprétation relève de ce que nous appelons un « synthétiseur virtuel ». Le polymorphisme est concentré dans une fonction générique appelée *synthesize*. Cette fonction est spécialisée par le nom du synthétiseur désiré, elle prend une liste d'*arrays* en filtrant toutes les instances qui ne sont pas des sous-classes de la classe abstraite associée au synthétiseur. Cette fonction traduit les *arrays* dans un format approprié : fichiers *orc* et *score* pour *CSound*, un fichier *SDIF* pour *Chant* (synthétiseur développé à l'Ircam), paquets *OSC* (protocole de communication développé au CNMAT à Berkeley) pour *Max/msp* ou *jMax* (Environnements de synthèse en temps réel développés à l'Ircam et par la société Cycling 74), etc. L'adjonction de nouveaux synthétiseurs au système sera réduite à la définition de nouvelles méthodes pour *synthesize*. La Figure 38 montre un exemple de la notion de synthétiseur virtuel. Le matériau de base vient d'une analyse d'un son de cymbale. Le côté gauche de la figure montre une synthèse utilisant *Chant*. Le côté droit renvoie les mêmes données d'analyse à *Csound*. Les résultats des deux synthèses sont presque identiques.

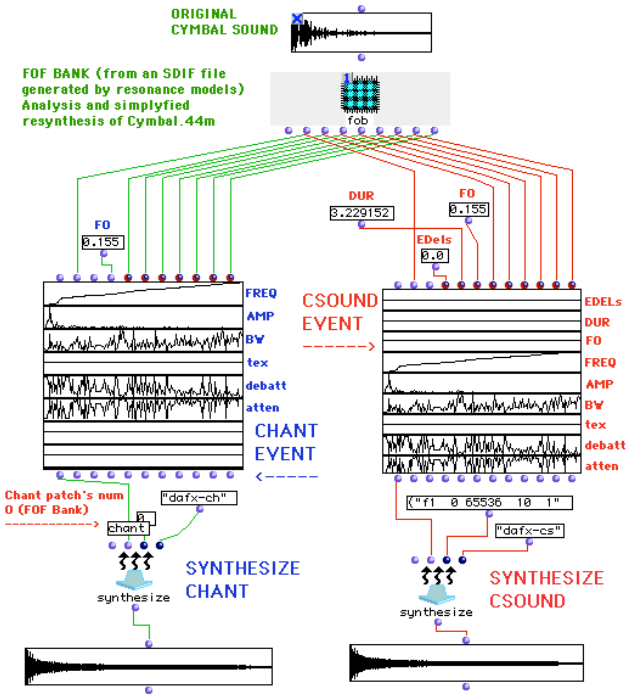


Figure 38. Création d'un array à partir d'un objet musical

Cependant, il est peu probable qu'un compositeur conçoive un processus de synthèse seulement en fonction d'un son unique. D'après notre expérience, un compositeur verra plutôt ce processus comme un modèle qui génère une famille de sons pouvant aussi être modifiée progressivement. De ce point de vue, la notion de son est remplacée par celle de potentiel sonore (voir section 2.3). Ainsi, les deux sons de la Figure 38 ne sont pas équivalents bien qu'il produisent le même résultat.

D'autres compositeurs ont utilisé OMChroma : Stephanie Schweiger dans "*verblich*" pour contre-basse et électronique *[om34]* ; Brian Ferneyhough dans *Stelae for the failed times [om27]*. Cependant, l'adoption des techniques par objets par les compositeurs reste une tâche à accomplir (ou à abandonner). Une des raisons de cette difficulté est le fait que les structures musicales utilisées par le compositeur, sauf celles qui sont proposées par OM, ne sont pas très complexes. Un compositeur ne construira pas une classe tempo, il utilisera plutôt une liste (e.g. '(60 4)) et la manipulera avec les fonctions *first* et *second* au lieu d'écrire des méthodes.

3.3 Programmation par contraintes.

L'application de la programmation par contraintes la plus célèbre en musique est sans doute celle de l'harmonisation automatique. Plusieurs travaux ont été réalisés autour de cette problématique [Ebc98], [TsA91], [Bal98], [PaR95]. Pour ceux qui désirent trouver un état de l'art sur l'harmonisation par contraintes, une référence inévitable est [PaR01]. Cependant, dans cette section, notre approche des contraintes est relativement différente. Comme on l'a déjà explicité dans le chapitre 1.7, la formulation d'un problème est pour nous aussi importante que sa résolution. C'est dans la perspective où le « que faire » est plus important que le « comment » que nous allons étudier la pertinence de l'approche par contraintes dans la composition.

Quatre systèmes de contraintes ont été intégrés dans OM : Situation [RuB98], OMRC [Lau93], Screamer [Sis93] et OMClouds [TAC03]. Dans cette partie, nous montrons un exemple de l'utilisation de OMRC pour deux sections de la pièce *Kalejdoskop* du compositeur Orjan Sandred. Bien que la partie harmonique de la pièce utilise aussi l'approche par contraintes, nous nous limiterons ici à développer sa partie rythmique. Pour une description détaillée de *Kalejdoskop* voir *[om31]*.

Dans *Kalejdoskop*, les contraintes rythmiques sont utilisées de deux manières différentes : soit les entités rythmiques sont le résultat d'un CSP (*Constraint Satisfaction Problem*), soit elles sont des mots pour un problème de contraintes qui va les concaténer en suivant certaines règles. Chacune des deux sections que nous considérons est une structure rythmique à trois voix. Ces trois voix sont liées par des contraintes sur les attaques de notes.

Dans la Figure 39, nous remarquons que chaque attaque de note dans la première voix a son équivalent dans la deuxième. Cette même contrainte d' « inclusion » est imposée aux deuxième et troisième voix.

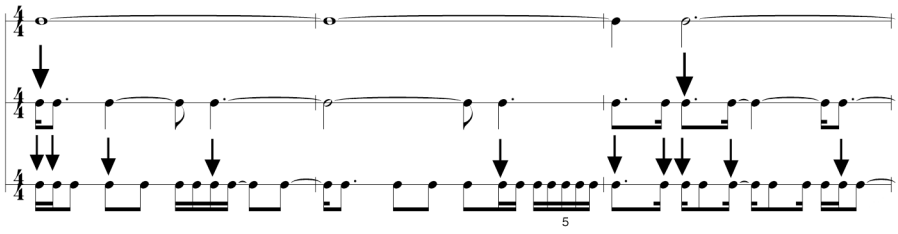


Figure 39 Structure rythmique de base

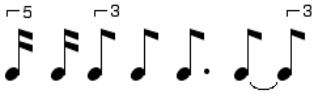
- Première section :

La première voix est contrainte à suivre une pulsation définie (9 noires dans notre exemple).

La deuxième voix prend ses valeurs rythmiques dans le domaine suivant :

Une contrainte est imposée de manière globale pour cette voix : au début de la pièce on favorisera les notes courtes et à la fin les notes longues, afin de donner l'impression d'un *ritardando*.

Pour la troisième voix, les valeurs rythmiques sont prises dans le domaine suivant :



Des heuristiques supplémentaires,pour cette voix, favorisent le choix de valeurs répétées, mais il existe une contrainte qui empêche une valeur rythmique d'apparaître plus de trois fois. Enfin, les triolets et quintolets ne peuvent pas commencer sur une pulsation.

- Deuxième section

La première voix de cette section est construite sur le même principe que celle de la première section. Pour les deux autres voix, les valeurs rythmiques sont cette fois, non plus des notes,

mais des phrases rythmiques qui devront, une fois superposées, respecter la contrainte d'inclusion entre les voix. La Figure 40 montre les valeurs possibles de ces phrases.

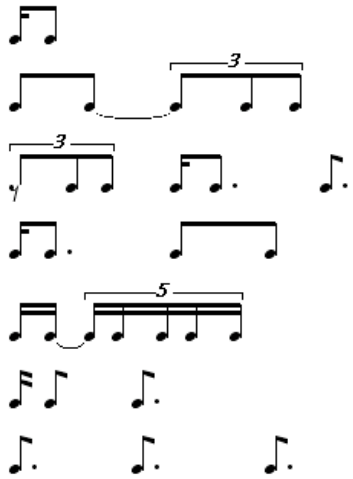


Figure 40 Domaine rythmique de la deuxième section.

Pour cette section, la contrainte de répétition de motifs a été supprimée car elle rendait le problème de l'inclusion de voix très difficile. Il existe cependant des exceptions à cette contrainte (Figure 41, troisième mesure). Cette violation de contraintes, faite « à la main », a pour intention de partager une séquence rythmique entre la troisième et la deuxième voix.

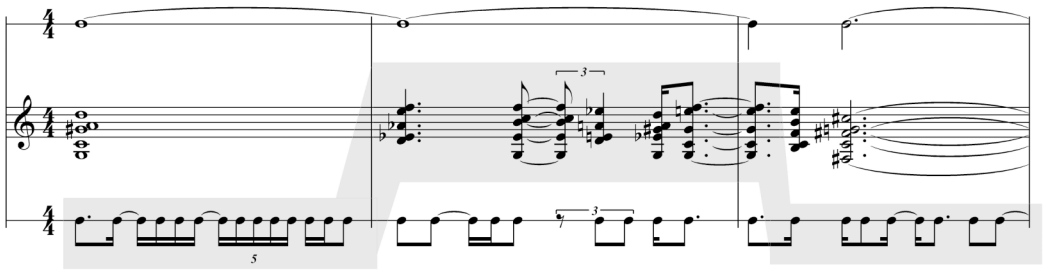


Figure 41. Exception de la contrainte d'inclusion entre les voix

Le patch de la Figure 42 montre l'implémentation dans OM de la structure rythmique de la première section. L'ensemble de contraintes est composé par des contraintes prédéfinies dans la librairie OMRC (e.g. *r-beat-subdiv*) et par des *patchs* écrits par le compositeur. Ils sont dans un mode graphique particulier qui permet de les utiliser en tant que foncteurs (e.g. *4thNotEq*).

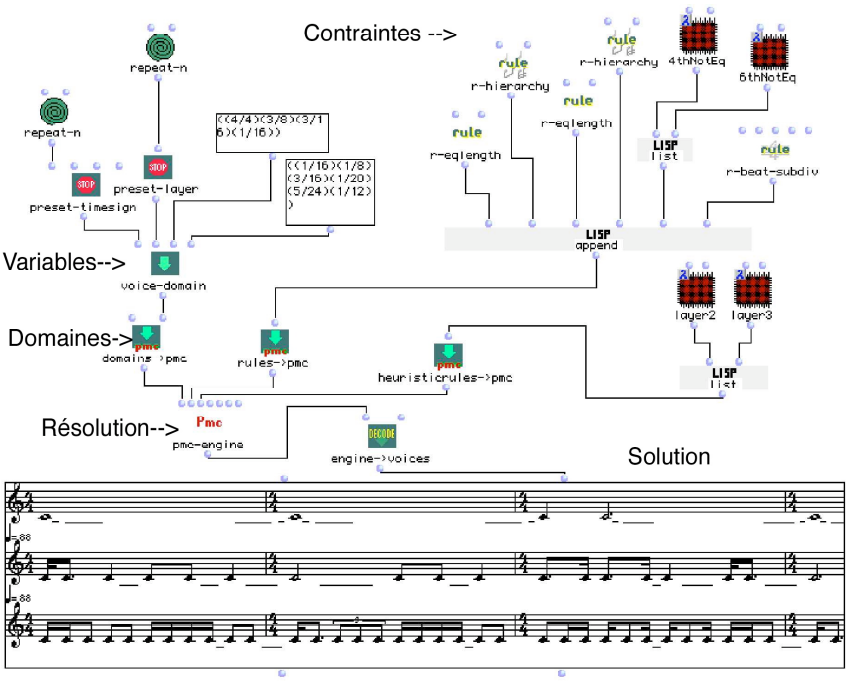


Figure 42. Patch OM pour la première section

Il est important de noter que les *patchs* de ce type ne sont pas construits d'un seul trait, ils sont le résultat d'un processus d'expérimentation qui, en même temps qu'il construit une solution potentielle, définit le problème de contraintes lui-même. Nous avons remarqué dans les TD du cursus de composition de l'Ircam que, généralement, le compositeur commence par un problème de contraintes assez simple, par exemple un tirage aléatoire dans un domaine fini. Ensuite, comme les solutions lui semblent triviales, il commence à rajouter des contraintes, (et dans la plupart des cas, jusqu'à formuler un problème qui n'a pas de solution). Et c'est à partir de ce moment qu'on peut parler de modèle : en premier lieu, le compositeur essaie de comprendre pourquoi son problème est « incohérent », il révise ensuite les contraintes, en enlève quelques-unes et en rajoute d'autres. C'est dans cette dynamique-là que les contraintes me semblent un outil pertinent pour la composition.

D'autres compositeurs utilisent les techniques de contraintes : Georges Bloch dans *Empreinte sonore de la fondation Beyeler [om6]* pour définir des textures harmoniques ; Mauro Lanza dans *Aschenblume [om17]* pour le contrôle de la synthèse par modèles physiques ; Jacopo Baboni-Schilingi pour mettre en rapport le contrôle global et local de sa pièce *Trois Profils [om4]*. Une compilation intéressante de l'application de contraintes dans les domaines harmonique, rythmique, mélodique, d'analyse musicale et d'interprétation, se trouve dans [Tru04].

Au vu des divers domaines d'application, fournir une bibliothèque de contraintes raisonnablement expressive pose certaines difficultés. On a en effet le choix entre deux extrêmes : écrire une bibliothèque adaptée à un problème bien précis et facile à utiliser (OMRC, par exemple), mais alors elle risque de ne servir qu'à un seul compositeur, ou bien construire un moteur de résolution qui accepte des contraintes générales (applicables à divers domaines), mais, d'une part, l'utilisateur devra faire un effort important pour écrire ces contraintes, et d'autre part on risque de perdre en efficacité : on ne peut pas demander au compositeur un trop grand travail d'optimisation dans l'expression des contraintes.

Le temps de calcul est un des soucis majeurs dans la programmation par contraintes. Comme tout utilisateur, le compositeur veut bien une solution dans un délai raisonnable. Attendre longtemps pour une réponse (éventuellement pour une non-réponse) ne semble pas le cas idéal, surtout s'il va la retravailler en fonction de ses propres critères esthétiques. En prenant en compte ces difficultés, Charlotte Truchet propose dans sa thèse [Tru04] l'utilisation d'un algorithme de recherche locale [CoD01] par améliorations successives du résultat. Cela permet d'abord de chercher des solutions approximatives, mais aussi d'afficher des solutions intermédiaires.

3.4 Un langage visuel multiparadigmes

Chacune des approches présentées dans ce chapitre offre un mode de pensée cohérent et unique. Elles permettent de modéliser une situation musicale particulière. Cependant, une composition présente plusieurs problématiques qui peuvent éventuellement faire appel à l'utilisation simultanée de plusieurs de ces paradigmes. Le but de la programmation multiparadigmes est de permettre au programmeur d'implémenter un programme en utilisant divers styles de programmation. Une énumération des diverses approches dans la programmation multiparadigmes peut se trouver dans [Spi94]. Deux familles peuvent être distinguées : celle qui définit un concept formel servant de base pour bâtir les divers paradigmes ; et celle qui tente de réunir divers paradigmes sur la base d'une approche « composant ». Sans prétendre donner une solution à l'intégration de paradigmes, nous utilisons notre langage visuel pour une telle intégration. Dans notre langage multiparadigmes, les *patchs* jouent un rôle intégrateur en visualisant en même temps fonctions, méthodes et prédicats. Etant donné le manque de formalisme et l'absence d'une réflexion approfondie à ce sujet, nous nous limiterons à donner un exemple de l'intégration du moteur de contraintes *Screamer* [Sis93] dans OM.

Screamer est un système de contraintes développé au MIT qui introduit des fonctions permettant le *backtracking*, ainsi qu'un moteur de résolution.

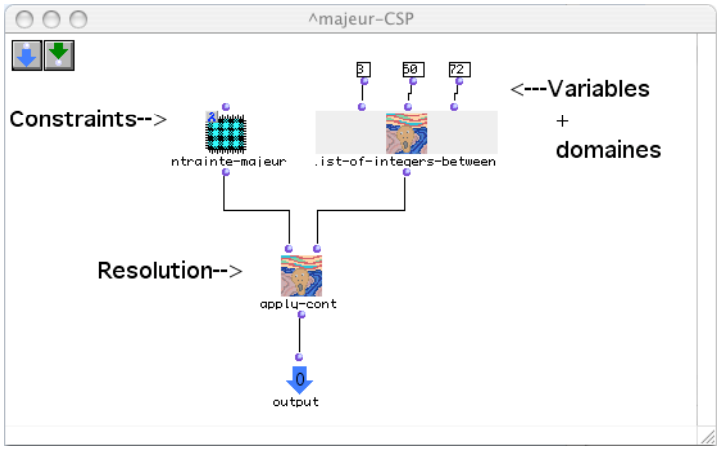


Figure 43. Définition graphique d'un CSP

Dans la Figure 43, la boîte *list-of-integers-between* définit une liste de trois variables. Chaque variable a pour domaine l'intervalle [60, 72]. Le patch *constrainte-majeur* est un prédicat résultant de la conjonction d'un ensemble de contraintes ($x_2 - x_1 = 5$ et $x_3 - x_2 = 2$). La boîte *apply-cont* définit le CSP à partir des variables et des contraintes.

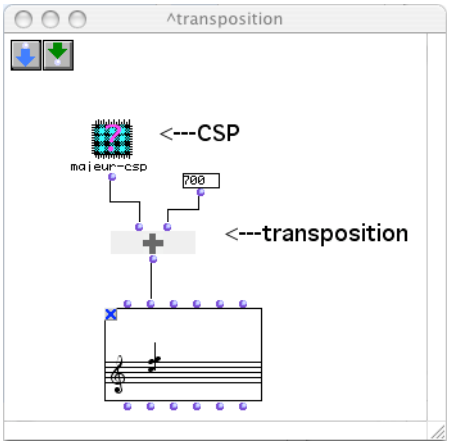


Figure 44. Utilisation du CSP dans un programme fonctionnel

Dans le *patch* de la Figure 44 le patch *constrainte-majeur* est inclut sous la forme de boîte. Puisqu'il s'agit d'un patch *Screamer*, la boîte est marquée par le symbole « ? ». Notre CSP fait ainsi partie d'un programme fonctionnel qui prendra le(s) résultat(s) du CSP et appliquera une transposition.

4. L'environnement OpenMusic

OpenMusic est né à la suite de PatchWork [Lau96], qui est un environnement de CAO développé par M. Larson, J. Duthen, C. Rueda, G. Assayag et moi-même. A la différence de PatchWork, qui peut être vu comme une interface graphique de Lisp, j'ai voulu faire avec OM un langage de programmation à part entière *[31]*. L'innovation principale d'OM est l'adaptation pour la CAO de divers styles de programmation : par objets, fonctionnelle, par contraintes et méta-programmation *[27]*. Un autre apport d'OM est la conception et l'implémentation des structures de contrôle graphiques parmi lesquelles les boucles, les conditionnelles, l'abstraction ou la récursion. Ce chapitre montre l'architecture d'OM et ses principales contributions.

4.1 Architecture

OpenMusic est implémenté en CommonLisp/CLOS [Ste98] sur Apple Macintosh. Le choix du langage obéit principalement aux raisons suivantes :

- c'est un langage fonctionnel et à objets puissant, bien défini, disposant d'un modèle formel solide [Cast98]
- CLOS et CommonLisp ont fait l'objet d'une normalisation internationale et ont un excellent degré de portabilité (des librairies graphiques orientées objet existent sur toutes les plate-formes)
- il existe une énorme quantité de savoir-faire en informatique musicale associée à ce langage
- le protocole de méta-objets (MOP [Pae93]) facilite le type d'extension que nous voulons réaliser.

Le MOP de CLOS contient une partie statique formée d'une hiérarchie de classes de méta-objets et une partie dynamique ou protocole de méthodes, qui permet la manipulation des méta-objets. Nous entendons par méta-objets les éléments du système objet sous-jacent à CLOS, telles que les fonctions génériques, les méthodes, les variables d'instances (*slots*) et les classes. Les deux tâches principales que j'ai réalisées pour l'implémentation d'OM ont été : la définition de sous-classes des classes de méta-objets et la définition ou redéfinition de méthodes pour ces nouvelles classes ou celles déjà existantes. OpenMusic constitue donc un enrichissement de la syntaxe et de la sémantique de CLOS. En outre, parce que certaines entités, telles que le patch ou la boîte, n'ont pas de correspondant dans CLOS (en effet, il n'y a pas de classe permettant de réifier la notion de programme ou d'invocation fonctionnelle dans CLOS), il m'a fallut définir des nouveaux méta-objets propres à OM. L'un des problèmes décisifs de cette implémentation a été le niveau de détail auquel on définit le protocole : en

général, un protocole très détaillé est peu modulaire. Ainsi, les changements doivent être effectués en plusieurs endroits, ce qui affecte la fiabilité. En revanche, un protocole peu détaillé rend délicat le choix de l'endroit où introduire les modifications *[8]*.

Partie statique

La Figure 45 illustre la partie statique d'OpenMusic. Les classes de méta-objets CLOS sont représentées par un rectangle gras.

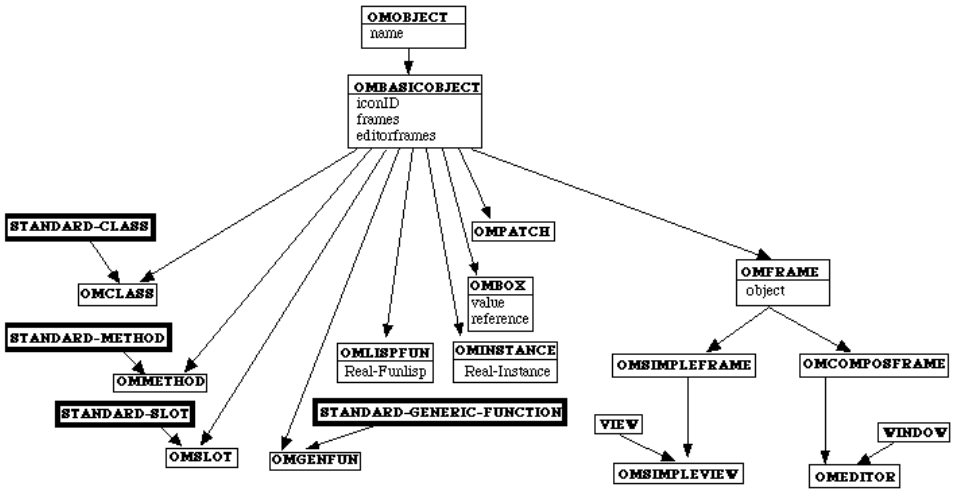


Figure 45 Hiérarchie de classes de méta-objets dans OpenMusic.

Cette hiérarchie de classes rend compte de tous les objets manipulables dans l'environnement visuel et dans le langage de programmation visuel d'OpenMusic. Les classes fondamentales du système objet visuel d'OpenMusic, *OMClass*, *OMMethod*, *OMSlot* et *OMGenFun*, héritent d'une part de la classe *OMObject* et d'autre part d'une classe fondamentale CLOS homologue. Les classes *OMPatch* et *OMBox* réifient respectivement les notions de patch et de boîte inexistantes dans CLOS. Les huit classes mentionnées, constituent les objets de calcul de notre langage (partie gauche de l'arbre d'héritage). La partie droite de l'arbre d'héritage implémente le paradigme visuel. On peut noter que la classe *OMFrame* hérite aussi de la classe *OMBasicObject*. Cela signifie que le paradigme visuel fait partie du langage, ceci étant fait dans l'intention de permettre à l'utilisateur de modifier et de contrôler ce paradigme.

Protocole dynamique

Les méta-objets dans OM sont composés par un ensemble d'éléments et optionnellement par des relations entre ces éléments. Par exemple, une classe est une liste ordonnée de champs, une fonction générique consiste en un ensemble de méthodes, un patch contient une liste de boîtes connectées, etc. Une fonction fondamentale dans notre protocole est donc la fonction *get–elements*, celle-ci retournant pour chaque *OMBasicObject* la liste de ses éléments, par exemple pour une OMClass nous aurons :

```
get-elements :   OMClass → liste de OMSlots
```

Le mécanisme de glisser–déposer est le principal outil d'édition dans OpenMusic. Une opération de glisser–déposer est définie comme une action entre une instance de la classe *OMsimpleView* appelée "source" et une instance de la classe *OMEditor* appelée "cible". Il existe un ensemble de prédicats *allow-drop* qui déterminent si l'opération de glisser–déposer est possi-

ble entre une paire (source , cible). Les fonctions *add-element* et *remove-element* ajoutés aux mécanismes de glisser–déposer et de délégation définissent le paradigme d'édition d'objets.

```
add-element :     OMClass × OMSlot → OMClass
remove-element :  OMClass × OMSlot → OMClass
```

Glisser une vue (source) dans la fenêtre d'un un éditeur (cible) produit en général l'application de la fonction générique *add-element* aux paramètres cible et source. Cette fonction est chargée d'effectuer les modifications visuelles dans l'éditeur ; elle doit aussi déléguer l'invocation de la fonction *add-element* au couple formé par les champs *object* de cible et de source. Cette dernière invocation est chargée de modifier l'objet de calcul visualisé par l'éditeur. Les fonctions suivantes, qui complètent le protocole, suivent le même paradigme :

`rename :` `Obj × name → Obj`

change le nom d'*Obj* par *name* .

`change-icon :` `Obj × iconID → Obj`

change l'icône d'*Obj* par *iconID*.

`select :` `Obj → Obj`

sélectionne *Obj*.

`unselect :` `Obj → Obj`

désélectionne *Obj*.

`moveobject :` `Obj × position → Obj`

change la position d'*Obj* à *position*.

`connect :` `SourceBox × i × TargetBox × j → Bool`

connecte la *i-ème* sortie de la boîte *SourceBox* à la *j-ème* entrée de la boîte *TargetBox*.

`box-value :` `Box × i → Obj`

évalue la *i-ème* sortie de la boîte *Box*.

4.2 Le langage visuel

Une formalisation graphique de la syntaxe et la sémantique de OM est donnée dans ma thèse de doctorat. Dans cette section, je présente sommairement la représentation graphique de divers paradigmes de programmation mentionnés.

Programmation fonctionnelle

La notion de base dans OpenMusic est le *patch*. Le patch réifie le concept d'algorithme. Les patchs contiennent des *boîtes* (icônes) et des *connexions* entre elles. Les boîtes représentent des appels fonctionnels, tandis que les connexions représentent des compositions fonctionnelles. Ces boîtes sont faites d'une icône et d'un ensemble ordonné d'entrées et de sorties. L'utilisateur peut déclencher une évaluation à n'importe quel point du graphe en cliquant sur une sortie de boîte. L'évaluation d'une boîte engendre une chaîne d'évaluations correspondant à l'exécution d'un programme. Le mécanisme d'abstraction d'un patch se fait en ajoutant des boîtes d'entrée et de sortie, représentées par des icônes de flèches. Une classe spéciale de boîtes permet l'application d'un patch dans un autre. Les entrées et soties de cette nouvelle boîte

correspondent aux boîtes d'entrée et de sortie dans l'abstraction. La Figure 46 montre l'abstraction du patch factorielle et son application dans un autre patch.

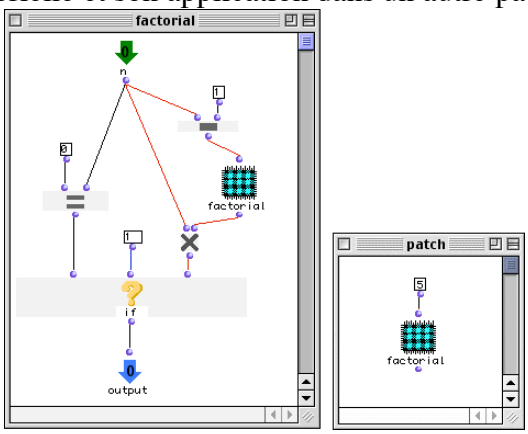


Figure 46. Définition du patch factorielle

Programmation par objets

OpenMusic offre une bibliothèque de classes pour la représentation et manipulation d'entités musicales. L'utilisateur peut étendre cette bibliothèque en créant de nouvelles classes, éventuellement en utilisant le mécanisme d'héritage. Il est aussi possible de créer de nouvelles fonctions génériques ou d'ajouter de nouvelles méthodes. La Figure Figure 47 présente une partie des classes musicales sous forme d'arbre d'héritage.

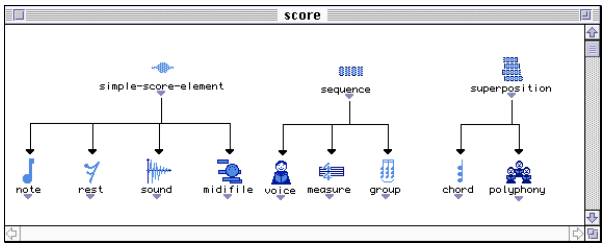
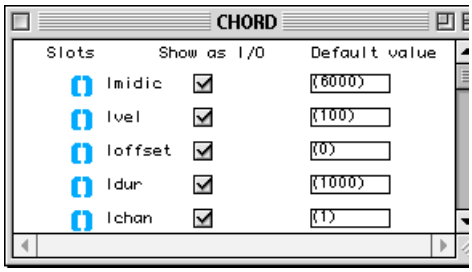


Figure 47. Arbre d'héritage des classes musicales

- Classes

Une classe est representée par une fenêtre contenant des slots. Voici, par exemple, l'éditeur de la classe *chord* (Figure 48) :



Lmidic. Une liste de midicents (hauteur des notes)
Lvel. Une liste de vélocités midi.
Loffset. Une liste d'offsets temporels (appoggiatures).
Ldur. Une liste de durées.
Lchan. Une liste de canaux midi.

Figure 48. Editeur de la classe chord

Graphiquement, une classe hérite d'une autre lorsqu'il y a une flèche de la super-classe vers la

sous-classe, comme illustré par la Figure 49.

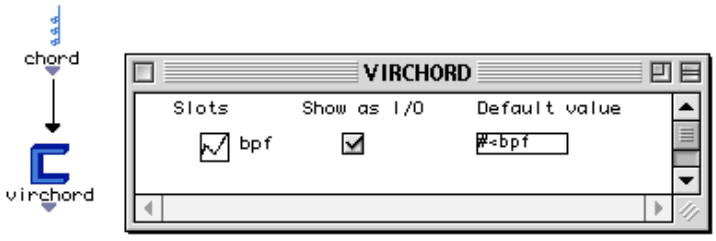


Figure 49. Exemple d'héritage

Une fois la nouvelle classe définie, nous pouvons créer par glisser-déposer de cette classe sur une fenêtre de patch, une boîte *factory* (Figure 50 en haut à gauche). Cette *factory* est en fait une boîte de création d'instances *virchord*. Elle possède tous les inlets et les outlets caractéristiques de la classe *virchord*. Sur l'inlet correspondant au champ *lmidic* est connectée une constante contenant une liste de hauteurs en midicents. A l'évaluation de la factory (par clic sur un des outlets), le mini-éditeur occupant le rectangle de la factory est mis à jour avec l'accord nouvellement créé. A partir de cette instance, un éditeur musical peut être ouvert par double-clic.

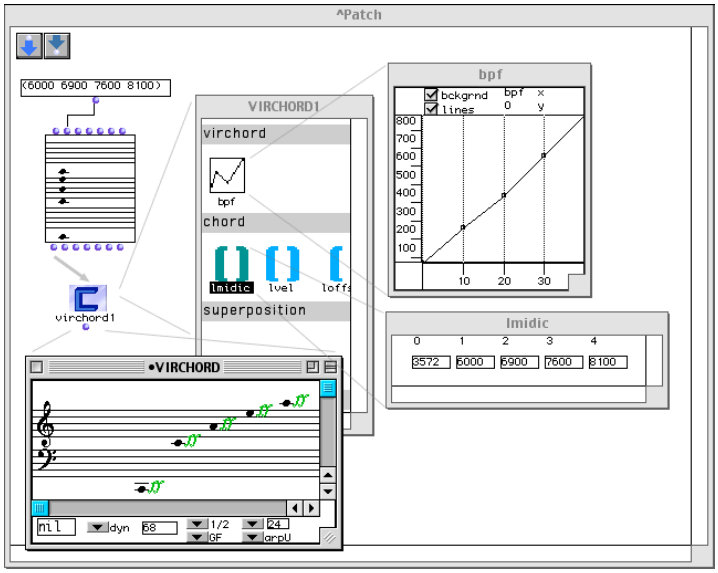


Figure 50. Création et édition d'une instance de Virchord

-Fonctions génériques, méthodes

La fonction générique prédéfinie *OM+* additionne 2 objets de type nombre ou liste de nombre (effectuant dans ce cas une opération vectorielle). Son éditeur-container (Figure 51 à gauche) montre la collection de méthodes correspondant aux combinaisons de ces types. Nous pourrons augmenter cette collection en lui ajoutant une méthode capable d'additionner deux accords *virchord* (Figure 51 à droite). Le container de la fonction générique OM+ est mis à jour avec la vue icônique de la nouvelle méthode, dont les types d'entrée sont indiqués par deux mini-icônes (en haut, fenêtre de gauche).

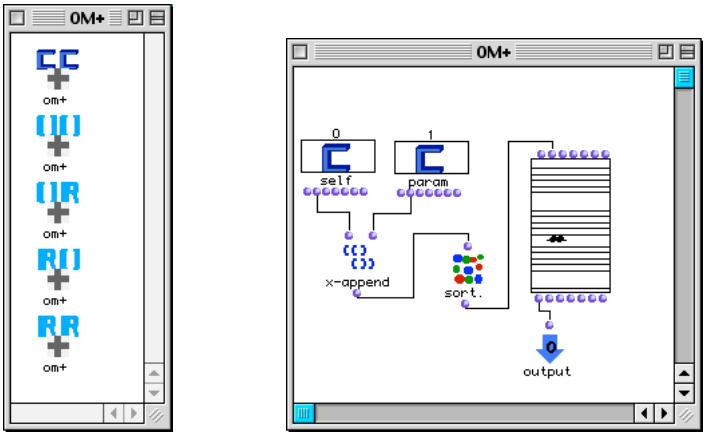


Figure 51. Création d'une méthode.

La Figure 52 montre un patch dans lequel deux *virchord* sont " additionnés " grâce à la fonction générique OM+, qui dispatche dans ce cas vers la nouvelle méthode. Le résultat, une instance de *virchord*, est directement branché dans l'entrée self d'une *factory* de la même classe. Il est à noter que l'utilisation directe de l'entrée self inhibe les autres entrées. Cela signifie que l'on désire initialiser la nouvelle instance non pas par des valeurs de champs, mais par la copie d'une instance existante (mécanisme appelé prototypage).

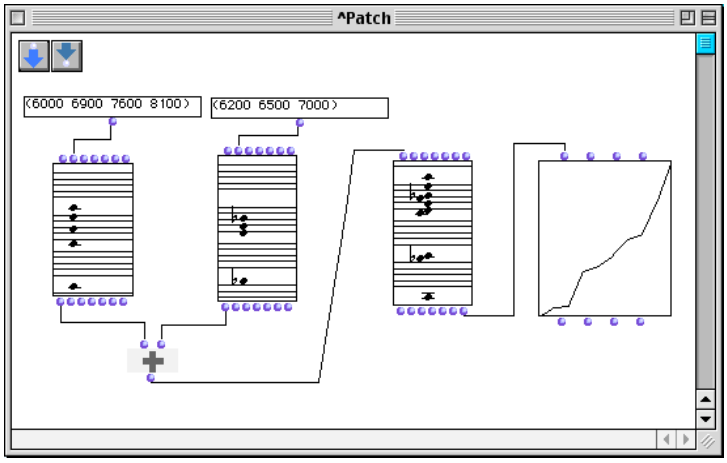


Figure 52. Addition de deux accords

Programmation par contraintes

Nous avons étendu les éditeurs d'OpenMusic par l'introduction de mécanisme de programmation de contraintes *[16]*. Illustrons à l'aide d'un exemple la définition graphique d'un CSP. Dans la Figure 53, nous posons une contrainte sur une séquence de douze notes sélectionnée auparavant. En double cliquant sur la représentation graphique du CSP, nous obtenons un éditeur permettant sa définition.

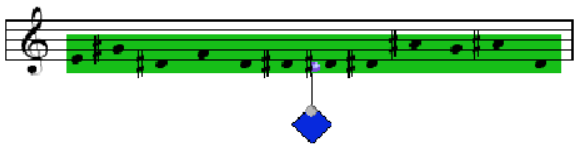


Figure 53. Contraintes dans une partition

Trois groupes de boîtes permettent de définir respectivement les variables, les termes et les contraintes. Dans l'exemple de la Figure 54, nous utilisons la boîte *insidevars* qui définit une variable pour chaque note de la séquence. Les variables sont liées aux attributs *midic* (représentant le pitch) de chaque note. Le domaine de chaque variables est l'intervalle entier des nombres compris entre 0 et 11.

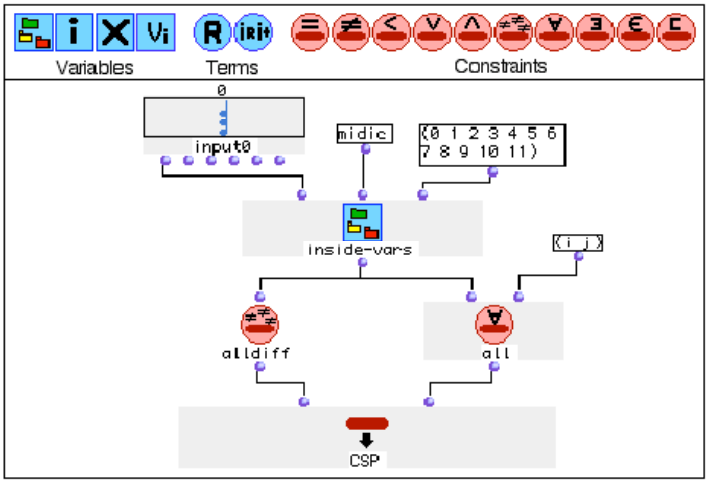


Figure 54. Définition graphique d'un CSP

Le problème comporte deux contraintes. La première force les variables à prendre des valeurs deux à deux différentes, elle est représentée par la boîte *alldiff*. La deuxième contrainte, représentée par la boîte *all*, impose que les intervalles entre notes consécutives de la série soient uniques. Ces deux contraintes permettent de définir une série dodécaphonique « tout intervalle ».

OpenMusic permet de résoudre un CSP en choisissant parmi plusieurs moteurs de contraintes. Un module de génération de code se charge de traduire le patch graphique en un CSP adapté au mécanisme de résolution choisi. Cette traduction n'a aucun effet sur la représentation graphique du CSP, et est de ce fait transparente pour l'utilisateur. Il existe trois moteurs des contraintes dans OM : Situation, Screamer ou OMClouds.

Situation est un moteur de contraintes par forward checking, conçu par Camilo Rueda [RuB98], et muni d'une interface graphique pour OM. L'une des originalités de Situation est la représentation par objets. Un objet est un ensemble de points et de distances. Il permet d'exprimer à la fois des rythmes ou des accords. Les contraintes portent sur les distances internes (entre points d'un même objet) ou externes (entre des points appartenant à des objets différents).

Screamer [Sis93] est un système implémenté par Mark Siskind qui ajoute à Common Lisp une forme d'indéterminisme via deux constructions, *either* et *fail*, introduisant des points de choix dans le langage. L'évaluation de l'expression (*either* $e_1 \ldots e_n$) évalue d'abord e_1, puis si l'évaluation de e_1 donne un *fail*, e_2, etc. Son intégration graphique dans OM a été exposée dans la section 3.4.

OMClouds [TAC03] est une librairie OM, implémentée par Charlotte Truchet, qui permet de modéliser et résoudre visuellement des problèmes de contraintes. OMClouds utilise un algorithme de recherche locale, la recherche adaptative, qui s'avère bien adapté à une utilisation musicale grâce à la possibilité d'édition des résultats partiels ou approchés pendant la résolution.

4.3 MOP Graphique

Nous avons utilisé le MOP de CLOS pour l'implémentation d'OM, comme cela a été précisé au chapitre 4.1. De la même manière que nous avons étendu CLOS lui-même en utilisant des techniques de méta-programmation, l'utilisateur peut étendre OM de manière graphique grâce au MOP visuel *[5]*, *[8]*. Dans cette partie, nous décrivons le MOP graphique d'OM et donnons certaines illustrations musicales. La Figure 55 montre une représentation graphique des parties statique et dynamique du MOP visuel.

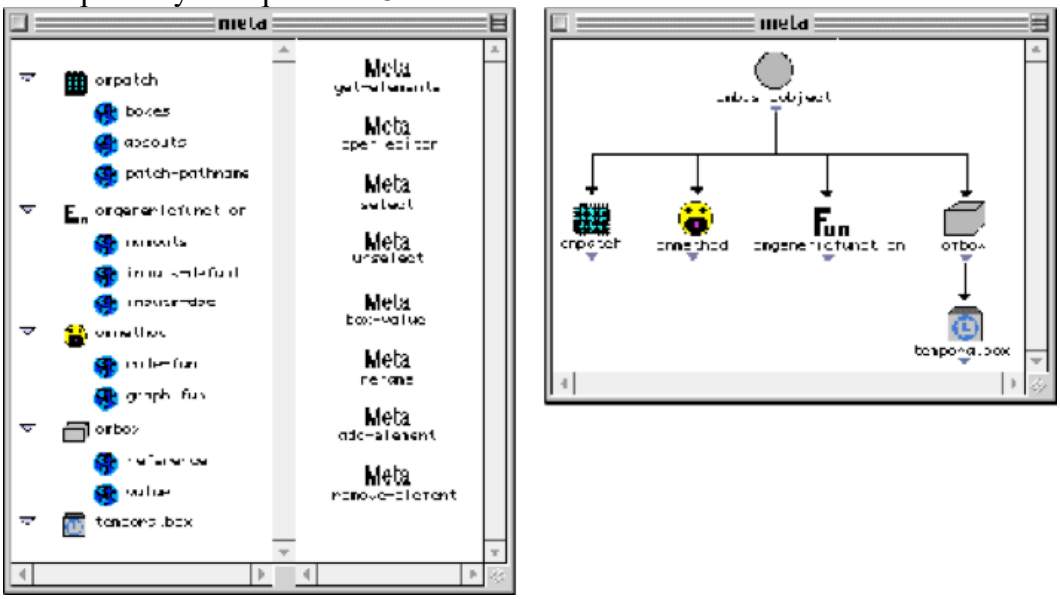


Figure 55. Représentation graphique du MOP

Sur la Figure 56 on voit la redéfinition de la fonction générique *box-value*, qui est appelée à chaque évaluation d'une boîte. Cette redéfinition se fait en ajoutant deux méthodes à la fonction générique. La première méthode, qui est définie avec le « qualifier » before, sélectionne graphiquement chaque boîte avant son évaluation. La seconde, avec le qualifier after, déselectionne la boîte après son évaluation. Cela nous permettra d'avoir une trace visuelle des évaluations successives.

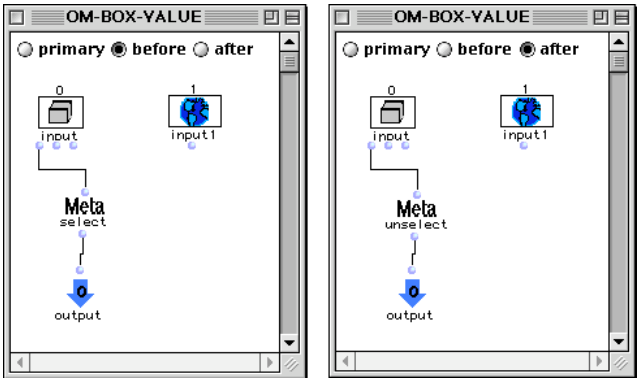


Figure 56. Redéfinition d'une fonction générique du MOP

Pour finir, nous allons montrer comment nous avons étendu les boîtes temporelles des maquettes en utilisant le MOP visuel. En effet, la réification graphique des boîtes temporelles permet d'accéder aux boîtes temporelles de la même manière qu'à une classe OM courante.

La Figure 57 montre un premier exemple de ces nouvelles possibilités. En ouvrant l'éditeur de la boîte temporelle, nous remarquons que le patch qui génère l'accord est construit en utilisant la boîte self, donc des informations de la boîte elle-même. Les sorties de la boîte *self* sont, de gauche a droite : la boîte elle-même, sa position et sa durée. Le processus associé à cette boîte construit un accord majeur dont les hauteurs sont transposées en prenant en compte la position de la boîte dans la maquette. Plus l'accord est avancé dans le temps, plus haut il sera transposé.

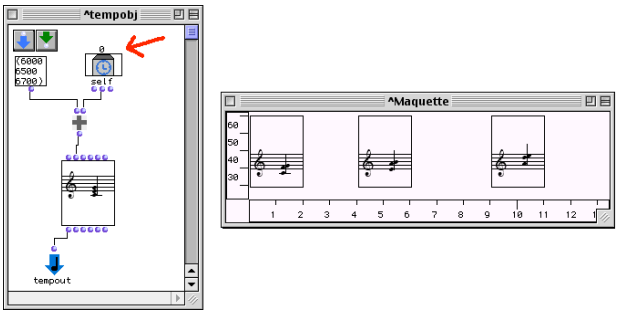


Figure 57. Calcul en fonction du temps

Les boîtes temporelles ont la possibilité de connaître les autres boîtes appartenant à la maquette. Dans la Figure 58, la boîte temporelle la plus en haut est envoyée à une deuxième boîte, laquelle modifie sa position temporelle en s'alignant avec la première boîte. Une évaluation de la maquette produit deux accords indépendants, mais contraints à commencer en même temps.

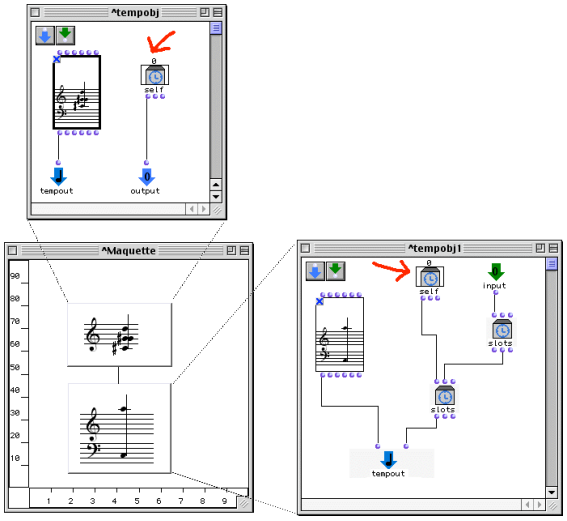


Figure 58. Position temporelle en fonction du calcul

Les maquettes proposent deux relations principales entre les boîtes : une relation de causalité et une relation temporelle. La causalité entre deux boîtes est définie par rapport au « temps d'évaluation ». L'évaluation d'une maquette suit le même paradigme que celui des patchs. Ce type d'évaluation étant très simple (nous n'avons pas d'évaluations en parallèle), il permet de définir entre les boîtes une relation d'ordre strict. De même, la règle temporelle de la maquette permet de comparer les objets temporels par rapport à leur « temps physique ». Les exemples des figures 9 et 10 montrent la combinaison de ces deux relations. Dans le premier exemple, le résultat du calcul dépend de la position de la boîte. Nous dirons que *le temps conditionne le calcul*. Pour le deuxième exemple, l'évaluation de la composition fonctionnelle des deux boîtes change leur position dans le temps. Dans ce cas, nous dirons que *le calcul conditionne l'organisation temporelle*.

4.4 Un standard pour la notation musicale

Les standards pour la représentation et la transmission de la notation musicale dans le domaine informatique continuent à susciter beaucoup d'intérêt de nos jours. Parmi les standards existants citons : [Mou96], [Gra95], [sch97], [HHR98]. Malgré les divers efforts, aucun standard ne s'est imposé à ce jour (contrairement au standard MIDI). Ceci peut s'expliquer par la diversité dans la finalité de leur utilisation : édition, gravure, performance, composition, saisie, analyse, visualisation. On constate que la variété de ces usages introduit un dualisme entre la représentation du contenu musical et sa mise en page.

Bien évidemment, la plupart des standards se situent entre ces deux extrêmes : presque tous ces formats mélangent de manière indistincte des informations musicales et des informations de formatage. Enfin, ce qui nous a paru faire le plus cruellement défaut dans la plupart des descriptions est le manque d'information hiérarchique du contenu lui-même. Très souvent les formats existants sont axés essentiellement autour de la description syntaxique et ne rendent pas compte d'une information fondamentale de la transmission de la musique, à savoir la sémantique.

Conscient de ces difficultés, dans *[10]* et *[14]*, j'ai défini un standard de notation destiné essentiellement aux compositeurs. Ce standard permet notamment la transmission de l'information sémantique.

L'idée principale est d'offrir une solution efficace aux applications dont la finalité est la manipulation et la construction des structures musicales. Notre approche vise à éliminer la prise en considération du formatage final concrétisé par la mise en page en vue de l'édition. Cette stratégie nécessite une méthodologie pour le rendu qui comblera le manque d'information concernant la pagination. De plus, dans la plupart des systèmes existants, les algorithmes de rendu ne sont pas publics. C'est pour les raisons suivantes que nous avons divisé le problème en deux parties complémentaires : premièrement, définition d'une représentation symbolique de structures rythmiques ; deuxièmement, mise à disposition d'un ensemble d'algorithmes pour le passage des représentations vers la notation sous forme de partition.

La représentation symbolique repose essentiellement sur une description hiérarchique du contenu musical telle qu'elle a été décrite dans le chapitre 3.1. Ceci permet une lisibilité de la structure hiérarchique prise dans son ensemble, offrant ainsi un format manipulable symboliquement (inversion, récursivité rétrograde, etc.) car le format est l'abstraction de l'objet rythmique même. Les algorithmes de rendu sont cependant indépendants du format de description. Leur intégration et leur implémentation auprès de divers environnements existants ou à venir est possible. Ceci nous permet à la fois d'intégrer des additions futures et d'ouvrir une dynamique pouvant élargir le champ descriptif du protocole même.

Concernant les hauteurs, je propose un système de notation appelé Omicron[1] pour des intervalles microtonaux. Voici par exemple les altérations pour le seizièmes de ton.

4.5 Applications dans la musique

OpenMusic est principalement disponible sur la plateforme Macintosh. Deux autres implémentations existent : un prototype tournant sous Linux, élaboré sous ma direction par Jose Diago et Gerardo Sarria dans le cadre du projet européen Agnula ; et une version pour Windows implémenté par Jean Bresson pour le projet Musiclab2, dont je suis responsable du développement. Le projet MusicLab2, réalisé en collaboration avec le Ministère de la Culture et de la Communication, vise à l'élaboration d'une approche informatique de la pédagogique musicale.

Outre son utilisation par de nombreux compositeurs, OM est une plateforme pour l'enseignement de la composition dans plusieurs universités et conservatoires : le Conservatoire National Supérieur de Musique et de Danse de Paris (CNSM) ; le Harvard University Departement of Music ; la Staatliche Hochschule für Musik und Darstellende Kunst Stuttgart ; le Computer Music Center de Columbia University. J'ai collaboré intensément à l'élaboration des supports de cours utilisés dans les programmes de ces institutions. L'un de ces cours, réalisé en collaboration avec Columbia University se trouve en version électronique sur le Web

[1] www.ircam.fr/equipes/repmus/mamux

à l'adresse http://music.columbia.edu/cmc/courses/OpenMusic/Course/Spring02. Mon activité pédagogique m'a également conduit à organiser des réunions de travail avec des compositeurs pour analyser les influences de la CAO pour la composition, notamment à Columbia University et au centre Temporeale de Florence en Italie. Un document dressant un état de l'art de la CAO a été produit à cette occasion (voir *[6]*).

Bien que sous licence GPL, OM est principalement distribué à travers le Forum Ircam, pour lequel je suis consultant. Le Forum Ircam est une communauté internationale de plus de mille professionnels de la musique (compositeurs, techniciens, musicologues, etc.) dont 829 utilisent OM. Je participe à deux réunions annuelles de trois jours pendant lesquelles les compositeurs montrent leurs œuvres et les utilisateurs expriment leurs besoins.

Plus qu'une simple plateforme pour la recherche et l'expérimentation, OpenMusic est donc un logiciel abondamment utilisé par les professionnels de la musique, tant les compositeurs pour leurs créations, que les musicologues pour l'analyse d'œuvres et les universités pour l'enseignement de la composition.

4.6 Utilisation dans la recherche

Outre la composition et l'enseignement, OpenMusic est une plateforme pour la recherche et l'expérimentation utilisée notamment :

- A l'université de Berlin par le groupe de Thomas Noll, pour l'analyse harmonique et rythmique : http://flp.cs.tu-berlin.de/~noll/
- Au MultiMedia Laboratory (MML), University of Zurich, dans le groupe de Guerino Mazzola pour l'analyse motivique : http://www.ifi.unizh.ch/mml/musicmedia/
- Au Centre Nationale de Creation Musicale CIRM Nice qui développe neurOMuse une bibliothèque OM destinée à l'étude et à l'application de reseaux neuromimétiques dans la création musicale : http://www.cirm-manca.org/cont.html
- A l'université de Cali dans le groupe Avispa de Camilo Rueda, l'objectif du groupe est de développer un modèle pour l'intégration des objets et des contraintes concurrentes dans un langage visuel : http://ingenieria.puj.edu.co/
- A l'université Ben Gurion en Israel, le groupe de Shlomo Dubnov en collaboration avec l'équipe représentations musicales de Gérard Assayag, utilisent OM pour modéliser le style musical : http://cse.cse.bgu.ac.il/~dubnov/
- A l'université d'Alicante en Espagne, OM est utilisé pour la reconnaissance de style http://www.dlsi.ua.es/~inesta/Inves/
- Le groupe Grupo de Pesquisa em Música, Musicologia e Tecnologia Aplicada (GMT) à UFPB, au Brésil, dirigé par le professeur Didier Guigue, utilise OM pour l'analyse musicale : http://lsd.dsc.ufcg.edu.br/~nazareno/

OpenMusic sert de plateforme d'expérimentation à plusieurs travaux de thèse de doctorat et de mémoires de DEA :

- Thèse de Benoît Meudic, université Paris 6, « Extraction automatique des structures musicales » . Avril 2004.
- Thèse de Charlotte Truchet, université Paris 7, « Contraintes, recherche locale et composition assistée par ordinateur ». Mars 2004.
- Olivier Lartillot-Nakamura, thèse de l'université Paris 6, « Un système d'analyse musicale computationnelle suivant une modélisation cognitiviste de l'écoute ». Mars 2004

- Moreno Andreatta, thèse de l'EHESS, « Méthodes algébriques en musique et musicologie du XXe siècle : aspects théoriques, analytiques et compositionnels ». Décembre 2003.
- Peter Hanappe, thèse de l'université Paris 6, « Design and implementation of an integrated environment for music composition and synthesis ». Décembre 1999.
- Nicolas Durand, DEA ATIAM, « Apprentissage du style musical et interaction sur deux échelles temporelles ». 30 juin 2003
- Jean Bresson, ESSI - Ecole Supérieure en Sciences Informatiques, Université de Nice - Sophia Antipolis, « Représentation et manipulation de données d'analyse sonore pour la composition musicale ». Septembre 2003.
- Emilie Poirson, DEA ATIAM, « Simulations d'improvisations à l'aide d'un automate de facteurs et validation expérimentale ». Juillet 2002.
- Benoît Mathieu, DEA MIASH, ENST, « Outils informatiques d'analyse musicale ». Juin 2002.
- Benoît Meudic, DEA ATIAM, « Modélisation de structures rythmiques ». Septembre 2000.
- Moreno Andreatta, DEA « Musique et Musicologie du XXème siècle » de l'EHESS, « La théorie mathématique de la musique de Guerino Mazzola et les canons rythmiques ». Juin 1999.
- Olivier Lartillot, DEA ATIAM, « Modélisation du style musical par apprentissage statistique : une application de la théorie de l'information à la musique ». Juillet 2000.
- Benoît Meudic, mémoire d'ingénieur de l'IIE, « Organisation rythmique dans un environnement d'aide à la composition ». Juin 1999.
- Jean-Louis Guénégo, DEA ATIAM, « Une application de résolution de contraintes en harmonie tonale » ; Juillet 1998.
- Olivier Delerue, DEA ATIAM, « Etude et réalisation d'opérateurs musicaux pour un environnement de composition assistée par ordinateur ». Juillet 1997.

4.7 The OM Composer's Book

Ce livre, que je suis en train de préparer, rassemble le témoignage de divers compositeurs sur la façon dont ils conçoivent l'utilisation de l'ordinateur dans leur acte compositionnel. Plus spécifiquement, chaque chapitre de ce livre fournit un exemple concret de CAO. Bien que ce livre réunisse un ensemble hétérogène de compositeurs et par conséquent des styles musicaux, le dénominateur commun de ces travaux est l'utilisation du logiciel OpenMusic pour leurs compositions. Après des années de pratique, je souhaite faire avec ce livre le point sur la situation actuelle de la CAO. Mais plutôt que de présenter de grands discours théoriques, ce livre préfère laisser la parole aux compositeurs. Chaque compositeur y présente une de ses oeuvres jouées, en mettant l'accent sur l'utilisation de OpenMusic dans son travail de composition, et plus généralement sur sa façon de concevoir la CAO.

La sélection des articles obéit principalement à un critère : la description d'une ou plusieurs pièces jouées. Comme conséquence directe, ce livre montre des résultats musicaux plutôt que des expériences. En plus de l'intérêt d'avoir le point de vue d'un nombre considérable de compositeurs sur la CAO, ce livre donne l'opportunité de comparer une grande diversité d'approches, de parcours artistiques et surtout de styles de composition. Ce livre pourra être consulté pour l'analyse musicale. En effet, on y pourra étudier les stratégies de production des musiques à travers les traces qu'elles laissent dans la mémoire d'un ordinateur. Grâce à la des-

cription des programmes informatiques utilisés dans la composition, le lecteur pourra centrer son attention sur divers pôles : la question du matériau ; la question de l'organisation du matériau (la structure et la forme) ; la question des raisons externes (perceptives, spéculatives, esthétiques...).

Ce livre est composé d'une introduction, suivie de 39 articles décrivant une démarche compositionnelle ou une pièce particulière d'un compositeur. Un CD audio avec des extraits musicaux illustrant chaque chapitre accompagnera le livre. La table des matières se trouve dans l'annexe A.

5. Limites de la CAO

Peut-on écrire un programme pour déterminer le style d'une œuvre musicale ? Comment faire un langage de CAO qui réifie les goûts musicaux, les désirs ou les fantasmes du compositeur qui l'utilise ? Que deviennent les intentions musicales d'un compositeur une fois qu'il doit les traduire pour la machine ? C'est à ce type de questions que l'on va s'intéresser dans ce chapitre. Conscients de la différence qui existe entre pouvoir effectuer une action et pouvoir la faire exécuter par un ordinateur, nous analysons ici les limites de l'utilisation des langages de programmation dans la composition.

Pour permettre aux compositeurs de faire des œuvres plus organiques nous devons intégrer à OM la notion de temps musical en tant que paramètre de composition. Une bonne partie de ce chapitre sera consacrée à ce problème.

4.1 S'attaquer aux problèmes difficiles

La plupart des problèmes intéressants en CAO sont NP-complets et/ou difficiles à formuler. Prenons, par exemple, une séquence d'accords quelconque et définissons un critère harmonique pour ordonner les accords afin de créer la meilleure progression harmonique [Ass00]. Nous pouvons modéliser ce problème sous forme d'un graphe complet dont les sommets sont les accords et les arcs sont étiquetés par le résultat de l'application d'une fonction de distance entre les sommets. Si la fonction de distance choisie possède un corrélat perceptif pertinent, nous aurons bien calculé par ce procédé un ordonnancement des points harmoniques qui maximise telle ou telle forme d'harmonie. En choisissant des critères simples comme le nombre de notes communes, ou la relation commune à une fondamentale (éventuellement virtuelle), ou la ressemblance de textures, ou le degré de rugosité, l'expérience montre une assez bonne adhésion des auditeurs aux résultats du calcul.

Malheureusement, le problème ainsi formulé constitue une instance du problème du voyageur de commerce, fameux pour être NP-Complet. Avec un grand nombre d'accords, les conséquences se font sentir très vite : il est impossible de calculer des solutions dans un temps raisonnable.

Pour s'attaquer à ce type de problèmes, il faut soit employer des techniques très spécialisées de recherche opérationnelles ou autre, afin de trouver les solutions optimales dans le meilleur temps possible, soit abandonner l'exigence d'optimalité, et employer des techniques incomplètes, comme des algorithmes statistiques, génétiques, neuronaux, etc.

Le noyau d'OpenMusic ne fournit pas de tels mécanismes aux compositeurs. En outre, les compositeurs ne sont en général pas désireux, ni capables, de programmer de telles techniques. De ce fait, une perspective pour l'amélioration de la puissance d'OM est de fournir des outils permettant d'utiliser ces techniques de façon intuitive.

Pour revenir au problème de construction de séquence harmonique, une solution ad'hoc utilisant des techniques probabilistes a été implémentée en OM. Cette méthode ne permet pas d'assurer que l'on trouve la solution optimale, en revanche, elle fournit des solutions de coût raisonnable (i.e. moins du double du coût optimal) en temps polynomial.

Outre les problèmes trop combinatoires pour être résolus par des techniques classiques, il existe des problèmes dont on ne trouve pas de formalisation informatique satisfaisante, même si la problématique semble bien définie. Par exemple, les problèmes traitant d'aspects physiques, comme la diffusion du son ou la représentation graphique d'une partition sur un écran, rentrent dans cette catégorie de problèmes.

Une dernière catégorie de problèmes difficiles à affronter est celle des problèmes qui ne sont même pas bien définis. Par exemple, le style musical est une notion très utile et à laquelle les musiciens font souvent référence sans toutefois pouvoir en donner une quelconque définition. Concrètement, pourrait-on écrire un programme prenant en entrée un fichier MIDI et donnant en sortie le style de la musique codée par ce fichier ? Si le style peut se définir par un ensemble de règles logiques, comme l'harmonie tonale, alors on peut écrire un tel programme, sans présumer de sa complexité (l'ordinateur est un excellent outil de classification) ; si le style est conçu comme dépendant d'une époque, d'une culture et d'influences historiques, alors le problème n'est pas une question de calcul mais d'esthétique.

En collaboration avec le GRIFA, groupe spécialisé en reconnaissance de formes et apprentissage, de l'Université d'Alicante, nous tentons en ce moment de classifier des styles musicaux. Pour cela, nous observons des compositeurs dans l'acte de composition, et en extrayons des informations statistiques. Nous détectons ainsi des patterns, ou des comportements récurrents, dans la manière de chaque compositeur.

4.2 Le temps musical

Le matériau premier de la musique étant le temps, nous devons disposer de fonctionnalités permettant de le manipuler. C'est ce qu'attend un compositeur de musique qui par sa pratique de la composition est habitué à ce travail sur le temps, et à le penser selon différents modes (cf. *en-temps* et *hors-temps* chez Xenakis [Xen81]). A notre époque, les discours sur le temps s'inscrivent dans un courant pluridisciplinaire qui va des sciences humaines, comme l'histoire, la littérature, la psychologie, la linguistique, entre autres, jusqu'à des sciences dites dures comme les mathématiques, la physique, en passant par l'économie, la médecine, etc. Il existe deux grands types d'approches par rapport à cette pluralité du temps : la tendance d'unification et la pluralisation du temps. Mais à mon avis, le problème ne consiste pas à savoir si le temps est un ou plusieurs. Si nous penchons pour l'optique unificatrice, il serait indispensable de profiter des travaux réalisés dans diverses disciplines afin d'élucider les mêmes mystères inhérents à la musique. Si nous favorisons l'optique d'irréconciliable divergence, il est clair que la nomination du temps dans n'importe quel domaine implique d'une manière ou d'une autre une relation étroite, soit par similitude soit par négation, avec le domaine musical. L'expression « temps musical » a divisé les esthéticiens et les musiciens du

XX^{ème} siècle en ceux qui nient son existence et ceux qui non seulement l'acceptent, mais placent le temps musical au cœur de la musique. Notre conception du temps musical est proche de celle de Francis Courtot : «...le temps dans la musique est différent du temps physique et philosophique. Le temps musical est plutôt en relation à la forme musicale qu'à un concept homogène et continu. »[Cou90].

Vers la fin du XIX^{ème} siècle, le temps commence à être considéré comme un paramètre de composition au même titre que la hauteur ou le rythme. C'est vers cette époque que l'expression « temps musical » fait son apparition, Ch. Koechlin est le premier à l'utiliser [Koe26] ; pour lui le temps musical est le résultat d'une organisation d'événements. L'argument qui permet d'établir l'autonomie du temps musical est l'impossibilité de réduire l'activité temporelle de la musique à une autre activité, que cette dernière soit gouvernée par le temps physique, psychologique ou philosophique. Cette impossibilité de réduction n'implique pourtant pas l'absence de relation, entre ces divers temps ; le temps musical a des propriétés quantitatives et qualitatives. Cette vision du temps musical comme résultat d'une organisation est décourageante à cause de la quantité de paramètres qui doivent être pris en compte, mais aussi à cause du caractère subjectif qu'elle impose. L'inclusion d'un « organisateur » du temps musical nous empêche de définir, pour une pièce, un énoncé axiomatique ; dans ce sens, nous ne croyons pas à l'existence d'un temps musical. Toutefois, il serait triste de clore nos recherches ainsi. La construction individuelle d'un temps par chaque auditeur nous semble une bonne idée pour continuer, cependant la question suivante semble logiquement s'imposer : si le temps musical est propre à chaque individu, comment peut-il être partagé par diverses personnes ? L'approche de G. Brelet [Eme98] semble bien convenir à la nôtre : le temps musical passe par les formes sonores, rythmiques et musicales. Ce sont les œuvres elles-mêmes qui engendrent le temps musicale, ce dernier étant inséparable du matériau et de l'organisation de la pièce. Voici donc une première prise de position : la pièce musicale est porteuse d'informations temporelles, sans cette affirmation, on imagine mal comment on pourrait utiliser la CAO en ce qui concerne le temps. Le statut du temps musical, en tant que paramètre de composition, implique non seulement son éventuelle manipulation (i.e. inversion), mais aussi la possibilité de le définir comme le résultat d'un calcul.

En informatique, deux groupes principaux d'approches peuvent être identifiés pour le traitement temporel : numérique et logique. La plupart des modèles temporels implémentés en CAO appartiennent au premier groupe. La majorité d'entre eux ont comme principale caractéristique l'irréversibilité et la linéarité ; ils sont donc applicables à un niveau très proche du temps physique. Le temps musical, pensé comme structure, présente une forte composante logique. Nous sommes intéressés par des modèles temporels qui nous permettent de traiter des figures comme la répétition, l'ambiguïté du choix ultérieur (*branching*) et, plus généralement, l'organisation des objets dans le temps, différente du modèle conventionnel « avant et après ». Les travaux en logique temporelle [BeL89] commencent à être explorés dans le cadre musical [PGR96], cependant l'appropriation de ces idées par le compositeur est presque inexistante. Plus complexe encore, est le déploiement de ce temps dans le temps physique. Sans une réalité sonore, le temps musical n'est qu'une hypothèse. Ce passage du logique au réel mérite une étude beaucoup plus approfondie.

6. Conclusions et perspectives

L'interaction entre la composition et l'ordinateur est une vieille histoire dont nous pouvons situer les premières expériences vers le milieu des années 50 *[6]*. Ces premiers travaux peuvent se résumer en trois étapes : la sélection du matériau, la combinaison aléatoire et la sélection parmi les résultats obtenus. Il est à noter que l'ordinateur n'avait aucun rôle à jouer dans la troisième phase, chose qui de nos jours reste vraie. La CAO « moderne », telle qu'on la voit émerger depuis le milieu des années 80, est conditionnée par un certain nombre de facteurs : la disponibilité d'ordinateurs personnels ; les progrès dans les interfaces graphiques ; la création de standards, dont la norme MIDI (*Music Instrument Digital Interface*) ; enfin, les progrès réalisés dans le domaine des langages de programmation, ces derniers constituent, selon nous, le facteur le plus important. L'utilisation d'un langage de programmation oblige le musicien à réfléchir au processus même de formalisation, lui évitant de considérer l'ordinateur comme une boîte noire qui lui impose ses choix.

Nous avons axé ce texte sur l'utilisation des langages de programmation pour la CAO. Nous avons proposé la figure d'un compositeur-programmeur et nous avons placé le modèle comme centre de l'interaction entre lui et le langage de programmation. La formalisation a été expliquée non comme une manière de standardiser la pensée musicale, mais au contraire comme un agitateur de la singularité. Ces prises de position ouvrent à leur tour beaucoup d'interrogations qui restent à étudier. N'importe quel langage peut-il servir à la composition ? Quelle est la place du calcul dans l'acte de composition ? Si le compositeur devient programmeur, doit-on reconsidérer le rôle du programmeur informaticien ? Le dernier chapitre a montré plusieurs limites de notre approche. Nos envisageons deux principales stratégies pour les travaux à venir : d'une part, l'approfondissement dans l'étude de la programmation et, d'autre part, la conception de nouveaux environnements de CAO combinant la programmation et d'autres techniques informatiques.

Approfondissement dans l'étude de la programmation
Nous étudierons la pertinence de l'adaptation de langages existants à nos besoins, principalement celle des langages concurrents que nous envisageons d'appliquer à l'écriture de la spatialisation. Nous étudierons également l'apport de la définition de nouveaux calculs. Le groupe Avispa et principalement Frank Valencia a développé récemment le calcul ntcc [NiV02], dont Camilo Rueda donne des application compositionnelles [DRF01] comme la possibilité de décrire de manière logique l'organisation temporelle d'une œuvre. L'intégration de plusieurs langages offre aussi de nouvelles possibilités d'expression. Actuellement, Carl Soleberg travaille sur la communication entre les logiciels Max et OM. Plutôt que de se

concentrer sur les aspects techniques, ce travail vise à donner une formalisation des systèmes temps-réel et à détecter les principales difficultés de la synchronisation de deux systèmes.

Conception de nouveaux environnements de CAO

Nous croyons que la programmation musicale peut s'enrichir d'autres techniques informatiques issues notamment de l'Interaction Homme Machine et du Génie Logiciel. La notion d'interaction sera au centre de ces travaux futurs. Peter Wegner énonce l'hypothèse que les programmes avec une boucle d'événements ne peuvent pas être formalisés par une machine de Turing [Weg97]. Ainsi, pour lui, les systèmes interactifs sont plus « puissants » que la machine de Turing. Aucune démonstration n'est donnée dans ces articles (le pouvoir d'expression est basé sur une thèse, pas sur un théorème). Même si ces idées semblent convaincantes, ma démarche ne vise pas à surpasser la machine de Turing, mais à détourner la machine de sa fonction de résolution de problèmes. Dans le chapitre 1, nous avons critiqué la vision de la CAO comme une discipline de la résolution de problèmes. Je conçois l'interaction comme un moyen de remise en cause dynamique du calcul. Un exemple de cette combinaison entre calcul et interaction a été présenté dans *[16]*. Nous avons défini un mécanisme de visualisation incrémentale des solutions pour l'algorithme de recherche adaptative que Charlotte Truchet applique à la satisfaction de contraintes : nous interrompons la résolution en certains points afin de visualiser la solution courante ; ceci se fait au détriment du temps de résolution, ce qui est un crime pour un chercheur en contraintes. Des couleurs associées aux variables montrent lesquelles violent au moins une contrainte. En outre, le compositeur peut arrêter le calcul à tout instant pour visualiser la solution courante, puis modifier la définition du problème (fixer des variables, ajouter des contraintes, etc.) en fonction de cette visualisation avant de reprendre le processus de résolution.

Bref, je m'intéresse en ce moment à l'interaction entre le compositeur et des processus d'évaluation. Dans le cas de la recherche adaptative, le problème est facile car l'algorithme est progressif et toutes les variables sont instanciées à n'importe quel moment de la résolution. En revanche, l'interaction avec l'évaluation d'une lambda-expression est plus complexe. En définissant un MOP (meta-object protocol) graphique pour OM *[8]*, nous avons voulu donner au musicien le contrôle sur le langage. Le MOP graphique n'a jamais été utilisé par les compositeurs-programmeurs : le « compositeur-designer » n'est qu'un rêve (pour l'instant).

Il faut noter que pour mettre en œuvre notre idée de la CAO, il ne suffit pas de créer des environnements informatiques et de les mettre à la disposition des compositeurs : un effort de diffusion et de pédagogie envers les compositeurs est nécessaire. Dans le cadre du projet *MusiqueLab 2*, nous nous sommes engagés en partenariat avec le Ministère de la Jeunesse, de l'Education et de la Recherche, et le Ministère de la Culture et de la Communication au renouvellement de l'approche pédagogique de la musique en prenant en compte les technologies informatiques. Plus généralement, ce projet vise à élargir nos recherches à d'autres enseignements artistiques. Une première étape de la collaboration entre l'Ircam et l'Education Nationale a été l'élaboration des *MusiqueLabs*, six applications explorant chacune un domaine particulier de la création sonore en s'appuyant sur des notions musicales fondamentales telles que hauteur, intensité, durée, rythme, temps, couleur. Les *MusiqueLab*s peuvent tout autant servir d'outils de découverte et de compréhension de processus spécifiques à la création musicale que d'ateliers d'invention et d'élaboration du sonore. Utilisés par le professeur, ils sont des outils d'expérimentation et de démonstration. Utilisés par l'élève, ils proposent des parcours de découverte du langage sonore (1200 établissements scolaires sont munis aujourd'hui de tels logiciels pédagogiques). Suite aux *MusiqueLabs*, les *MusiqueLabs 2* visent à une progression dans la complexité d'utilisation de la part de l'élève en étendant les possibilités de

composition bien au-delà du simple montage, vers un outil basé sur la *maquette* d'OM. Prenons par exemple une application pour la génération de matériaux harmoniques à partir d'entités élémentaires : accords, sons, fichiers Midi. Présentée sous forme de maquette, elle permettra des expériences en changeant, soit les entités élémentaires soit les programmes visuels qui implémentent les relations entre elles. Les *MusiqueLabs2* seront principalement accessibles sous la forme de maquettes OM où l'on pourra toujours accéder aux algorithmes visuels sous-jacents, pour programmer soi-même l'outil. Les maquettes permettront à l'élève l'application des transformations musicales dans le temps, et d'établir des dépendances entre ces transformations tout en disposant d'un rendu à partir de fichiers son ou MIDI. Dès 2005, ce programme s'appliquera de façon expérimentale à l'échelle régionale. A la demande du Ministère de la Culture, l'Ircam va s'engager à son application dans des écoles et conservatoires de musique.

Actuellement, OpenMusic est enseigné dans plusieurs universités et conservatoires et utilisé par un nombre croissant de compositeurs et de musicologues. L'utilisation de plus en plus répandue d'OM par des compositeurs de musique contemporaine a relativisé la complexité de certaines techniques en vogue depuis la révolution dodécaphonique et son héritage. Lorsqu'un ordinateur peut calculer très rapidement toutes les variantes combinatoires issues d'un formalisme donné, par exemple le formalisme sériel, la valeur musicologique ne peut plus venir simplement de ce principe formel, mais de son adéquation réelle à la perception. Corrélativement, l'ordinateur permet d'explorer des champs expérimentaux d'une complexité réelle et qui étaient inaccessibles auparavant, comme par exemple le champ du timbre dans sa représentation spectrale. Parallèlement, OpenMusic a favorisé la démarche expérimentale qui permet de soumettre un très grand nombre d'instances musicales issues d'un formalisme au test de la musicalité. Plus encore, elle autorise l'expérimentation sur les formalismes eux-mêmes qui peuvent être testés et en retour modifiés ou abandonnés s'ils ne tiennent pas leurs promesses. Enfin, OpenMusic propose un renouvellement de l'idée de notation qui devient dynamique, incluant les mécanismes génétiques de l'oeuvre. La partition constitue d'une certaine manière sa propre analyse, ce qui est déjà le germe d'un bouleversement des habitudes musicologiques. Nous avons essayé avec OpenMusic de construire un langage qui encapsule de manière consubstantielle la notion de notation, à savoir notation du résultat (une partition musicale) mais aussi notation du processus aboutissant à ce résultat (programme visuel).

7. Bibliographie

[Ass93] Gérard Assayag. CAO : vers la partition potentielle. Les Cahiers de l'Ircam: La composition assistée par ordinateur 1(3), juin 1993.

[Ass98] Gérard Assayag. Musique, Nombre, Ordinateur. Fondation Gulbenkian. 1998.

[Ass00] Gérard Assayag. De la calculabilité à l'implémentation musicale. Deuxième saison du Séminaire MaMuX. www.ircam.fr/equipes/repmus/mamux.

[And03] Moreno Andreatta. Méthodes algébriques dans la musique et la musicologie du XXème siècle : aspects théoriques, analytiques et compositionnels. Thèse de l'Ecole des Hautes Etudes en Sciences Sociales. Paris, 2003.

[Bal96] Mira Balaban. The Music Structures Approach to Knowledge Representation for Music Processing. Computer Music Journal, 20(2), pp. 96-111, 1996.

[Bal98] Ballesta Philippe. Constraints et objets, clefs de voûte d'un outil d'aide à la composition. Ed. Hermes, 1998.

[BeL89] H. Bestougeff et G. Ligozat. Outils logiques pour le traitement du temps. Masson. Paris, 1989.

[Bre03] Jean Bresson. Représentation et manipulation de données d'analyse sonore pour la composition musicale. Rapport de stage ESSI - Université de Nice Sophia Antipolis, 2003.

[CoD01] P. Codognet and D. Diaz. Yet Another Local Search Method for Constraint Solving. Proceeding of the AAAI Symposium "Using Uncertainty in Computation", T. Walsh, C. Gomes (Eds.), Cape Cod, AAAI Press October 2001.

[Cou90] Francis Courtot. CARLA : Knowledge acquisition and induction for computer assited composition. Rapport de recherche, Ircam, 1990.

[Cou93] Francis Courtot. Entre le décomposé et l'incomposable. Les cahiers de l'Ircam 1(3) : La composition assistée par ordinateur 1(3). Paris juin 1993.

[DRF01] Juan F. Diaz, Camilo Rueda, Frank D. Valencia et al. Integrating Constraints and Concurrent Objects in Musical Applications: A Calculus and its Visual Language. Constraints Journal 6(1), 2001.

[Ebc98] Kemal Ebcioglu. An expert system for harmonizing chorals in the style of J.-S. Bach. Notes in Computer Science, 1998.

[EcG94] Eckel, G., González-Arroyo, R. Musically Salient Control Abstractions for Sound Synthesis. Proceedings of the 1994 International Computer Music Conference Aarhus,1994.

[Eme98] Eric Emery. Temps et musique. L'age de l'homme. Lausanne, 1998.

[GHJ95] E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design patterns, elements of reusable object-oriented software. Addison-Wesley publishing company. 1995.

[Gra97] Grande, Cindy. The Notation Interchange File Format : A Windows - Compliant Approach. In: E. Selfridge-Field, ed. Beyond MIDI - A Handbook of Musical Codes. Massachusetts : MIT Press. , 1997.

[Han95] Peter Hanappe. Integration des représentations temps/fréquence et des représentations musicales symboliques. Rapport de DEA ATIAM, Ircam - Université Paris VI. Paris, 1995.

[HHR98] H. H. Hoos, K. A. Hamel, K. Renz, J. Kilian. The GUIDO Music Notation Format - A Novel Approach for Adequately Representing Score-level Music. In ICMC'98 Proceedings, p.451-454.

[Koe26] Koechlin Ch. Le temps et la musique. La revue musicale No 3. Paris 1926.

[Lau96] M. Laurson. PATCHWORK: A Visual Programming Language and some Musical Applications. Helsinki : Sibelius Academy, Helsinki, 1996.

[Lau93] Laurson M. PWConstraints. In Haus G. and Pighi I. (eds), X Colloquio di Informatica Musicale, 332-335, Associazione di Informatica Musicale Italiana, . Milano 1993.

[Lev92] Pierre Lévy. De la programmation considerée comme un des beaux-arts. Ed. la découverte. Paris, 1992.

[Lus86] Pierre Lusson. Place d'une théorie générale du rythme parmi les théories analytiques contemporaines. Analyse Musicale n 2, pp. 44-51, 1986.

[NiV02] Mogens Nielsen and Frank D. Valencia. Temporal Concurrent Constraint Programming: Applications and Behavior. Chapter in W. Brauer, H. Ehrig, J. Karhumäki, A. Salomaa (Eds.). 2002.

[Mal01] Mikhail Malt. Les mathématiques et la Composition assistée par Ordinateur (Concepts, Outils, Modèles), thèse, EHESS. Paris, 2001.

[Mar72] J. D. Markel, The SIFT algorithm for fundamental frequency estimation, IEEE Trans. Audio and Electroacoustics, vol. 20, pp. 367--377, 1972.

[Meu99] Benoit Meudic. Organisation rythmique dans un environnement d'aide à la composition. Rapport DEA Atiam. 1999.

[Meu04] Benoit Meudic. Extraction automatique des structures musicales. Thèse de doctorat à l'Université de Paris 6, Paris 2004.

[Mou96] Stephen R. Mounce. A Brief Discussion of Standard Music Description Language. ISO/IEC Draft International Standard 10743, 1996.

[OrE96] Orfali R., Harkey D., Edwards J. The essential Distributed Objects. Wiley ed. 1996.

[OrF97] Yann Orlarey, Dominique Fober et Stéphane Letz. Elody : a Java+MidiShare based Music Composition Environment. Proceedings of the ICMC 97, Thessaloniki, 1997.

[Paep93] Andreas Paepcke. Object-Oriented Programming - The CLOS Perspective. MIT Press, 1993.

[PGR96] François Pachet, Geber Ramalho and Jean Carrive. Representing Temporal Musical Objects and Reasoning in the MusES System.JNMR 1996 - Issue Number 3.

[PaR01] Francois Pachet, Pierre Roy. Musical harmonization with constraints : a survey. Constraints 2001.

[PaR95] Francois Pachet, Pierre Roy. Integrating constraint satisfaction techniques with complex object structures. British Computer Society Specialist Group on Expert Systems. 1995.

[Rod85] X. Rodet, Y. Potard, J.-B. Barrière. CHANT - de la synthèse de la voix chantée à la synthèse en général. Ircam Paris, 1985.

[RoC85] Rodet X. et Cointe P. FORMES Composition et ordonnancement de processus. Rapports de recherche No 36, Ircam, Paris, 1985.

[RuB98] Camilo Rueda et Antoine Bonnet. Un langage visuel basé sur les contraintes pour la composition musicale. Chapitre 5 dans Recherches et applications en informatique musicale. Hermes, Collection Informatique Musicale, Paris, 1998.

[sch97] Bill Schottstaedt. Common Music Notation. In : Beyond MIDI, The handbook of musical codes. Eleanor Selfridge-Field ed. MIT press, Cambridge Masssachusetts, 1997.

[Sis93] Jeffery Mark Siskind and David McAllester. Nondeterministic Lisp as a Substrait for Constraint Logic Programming. AAAI-93, pages 133-138.

[Spi94] Diomidis D. Spinellis. Programming Paradigms as Object Structuring Mechanism for Multiparadigm Programming. PhD thesis, Science, Technology and Medicine, London, 1994.

[Ste98] Guy L; Steele. Common LISP The language, second edition. Digital Press, USA, 1998.

[SwW00] D. Schwarz, M. Wright. Extensions and applications of SDIF. Proc. ICMC 2000 Berlin, 2000.

[TAC03] Charlotte Truchet, Gérard Assayag, Philippe Codognet. OMClouds, petits nuages de contraintes dans OpenMusic. Proc. JIM03. Montbeliard, 2003.

[Tan01] A. Tangian. The Sieve of Eratosthene for Diophantine equations in integer polynomials and Johnson's problem. Discussion paper No. 309, FernUniversity of Hagen, 2001.

[Tau02] H. Taube. Common Music Reference Manual. University of Illinois, Illinois 2002.

[Tru04] Charlotte Truchet. Contraintes, Recherche Locale et Composition Assistée par Ordinateur. Thèse d'Informatique à l'Université Paris 7. Paris 2004.

[TsA91] C. Tsang, M. Aitken. Harmonizing music as a discipline of constraint logic programming. ICMC, 1991.

[Vuz91] D.T. Vuza. Supplementary Sets and Regular Canons I, Perspectives of New Music, 29(2). 1991.

[Weg97] Peter Wegner. Why Interaction Is More Powerful Than Algorithms, Communications of the ACM., May 1997.

[Xen81] Xenakis I. Musiques formelles. Stock Musique, Paris, 1981.

Publications personnelles :

Revues à comité de lecture

[1] Formal aspects of Iannis Xenakis' Symbolic Music: a computer-aided exploration of some compositional processes. Carlos Agon, Moreno Andreatta, Gérard Assayag,Stephan Schaub. Journal of New Music Research (to appear 2004).

[22] Elementi di teoria matematica della musica e problema della classificazione di canoni regolari complementari di categoria massimale. Moreno Andreatta, Carlos Agon. Il Monocordo, Vol 7/8, Salerno, 2000.

[24] Computer Assisted Composition at Ircam : PatchWork & OpenMusic. Gérard Assayag, Camilo Rueda , Mikael Laurson, Carlos Agon, O. Delerue. Computer Music Journal 23:3, MIT Press, 1999.

Chapitres dans un livre

[2] Anatol Vieru : formalisation algébrique et enjeux esthétiques. Costin Cazaban, Moreno Andreatta, Carlos Agon, Dan Tudor Vuza. «Le livre du Séminaire MaMuPhi ». L'harmattan Ed. (to appear 2004).

[3] Mixing Visual Programs and Music Notation. « Perspectives of Mathematical Music Theory ». G. Mazzola and T. Noll Ed., epOs Music, Universität Osnabrück (to appear 2004).

[6] Outils informatiques pour la composition musicale assistée par ordinateur: un état de l'art. Carlos Agon et Gérard Assayag, « Book of articles and essays of PRISMA », Euresis Edizioni, Milano 2003.

[8] Object-Oriented Programming in OpenMusic. « Topos of Music ». Mazzola G. Birkhäuser Verlag Ed. Zurich 2003.

[18] OpenMusic, un environnement de programmation visuelle pour la composition. Gérard Assayag, Carlos Agon. In Encyclopédie pour l'Ingénieur, Informatique Musicale, F. Pachet Ed., Hermès, Paris 2001.

Conférences internationales

[4] Implementing algebraic methods in OpenMusic. Moreno Andreatta, Carlos Agon. Proc. ICMC, Singapore 2003.

[5] OM: A Graphical extension of CLOS using the MOP. Carlos Agon et Gérard Assayag. Proc. ICL03, New York, 2003.

[10] Representation and Rendering of Rhythmic Structures.
Carlos Agon, Karim Haddad et Gérard Assayag. WedelMusic Darmstadt, IEEE Computer Press. 2002.

[11] Tiling problems in music composition: Theory and Implementation. Moreno Andreatta, Carlos Agon, Emmanuel Amiot. Proc. ICMC, Göteborg 2002.

[16] A Constraint Programming System for Music Composition, Preliminary Results. Charlotte Truchet, Carlos Agon, Philippe Codognet. Workshop Musical constraints CP-01, Chypre 2001.

[21] High Level Musical Control of Sound Synthesis in OpenMusic. Carlos Agon, Marco Stroppa, Gérard Assayag. Proc. ICMC 2000, Berlin, I.C.M.A., San-Francisco, 2000.

[27] Objects, Time and Constraints in OpenMusic. Carlos Agon, Gérard Assayag, Olivier Delerue, Camilo Rueda. Proc. ICMC 98, Ann Arbor, Michigan, I.C.M.A., San-Francisco, 1998.

[28] An Object Oriented Visual Environment For Musical Composition. Gérard Assayag, Carlos Agon, Joshua Fineberg, Peter Hanappe. Proceedings of the ICMC 97, Thessaloniki, I.C.M.A., San-Francisco, 1997.

[31] OpenMusic Architecture. Gérard Assayag, Carlos Agon. Proceedings of the ICMC 96, Hong Kong, I.C.M.A., San-Francisco, 1996.

[32] Kant: a Critique of Pure Quantification. Carlos Agon, Gérard Assayag, Joshua Fineberg, Camilo Rueda . Proceedings of the ICMC 94, Aarhus, I.C.M.A., San-Francisco, 1994.

Conférences nationales

[7] Formalisation algébrique des structures musicales à l'aide de la *Set-Theory* : aspects théoriques et analytiques. Moreno Andreatta et Carlos Agon. Actes du JIM 2003, Montbeliard, 2003.

[9] Programmation Visuelle et Editeurs Musicaux pour la Composition Assistée par Ordinateur.Carlos Agon, Gérard Assayag. IHM 02, Poitiers. ACM Computer Press. 2002.

[12] OmChroma ; vers une formalisation compositionnelle des processus de synthèse sonore. Marco Stroppa, Serge Lemouton, Carlos Agon. Proc Jim Marseille 2002.

[13] Contrôle de la spatialisation comme paramètre musical. Gilbert Nouno, Carlos Agon. Proc Jim Marseille 2002.

[14] Représentation et rendu de structures rythmiques. Carlos Agon, Karim Haddad, Gérard Assayag. Proc Jim Marseille 2002.

[15] Analyse et implémentation de certaines techniques compositionnelles chez Anatol Vieru. Moreno Andreatta, Carlos Agon, Dan T. Vuza. Proc Jim Marseille 2002.

[17] Recherche adaptative et contraintes musicales. Charlotte Truchet, Carlos Agon, Gérard Assayag. Journées francophones de programmation logique et programmation par contraintes. Paris 2001.

[19] CAO et contraintes. Charlotte Truchet - C. Agon - G. Assayag - P. Codognet Proc. JIM 2001, Bourges.

[20] The Geometrical Groove: rhythmic canons between Theory, Implementation and Musical Experiment Andreatta Moreno, Noll Thomas, Agon Carlos, Assayag Gérard Proc. JIM 2001, Bourges

[23] OpenMusic et le problème de la construction de canons musicaux rythmiques. Moreno Andreatta, Carlos Agon, Marc Chemillier. Proceedings JIM 99. CeMaMu. Issy-les-moulineaux. 1999.

[25] OpenMusic + MusicSpace = OpenSpace. Olivier Delerue, Carlos Agon. Proceedings JIM 99. CeMaMu. Issy-les-moulineaux. 1999.

[26] Etude et réalisation d'opérateurs rythmiques dans OpenMusic, un environnement de programmation appliqué à la composition musicale. Olivier Delerue, Gérard Assayag, Carlos Agon, Proceedings JIM 98, La Londe les Maures, 1998.

[29] Problèmes de notation dans la composition assistée par ordinateur. Gérard Assayag, Carlos Agon, Joshua Fineberg, Peter Hanappe. Actes des Rencontres Musicales Pluridisciplinaires, Musique et Notation. GRAME. Lyon 1997.

[30] Problèmes de quantification et de transcription en composition assistée par ordinateur. Gérard Assayag, Peter Hanappe, Carlos Agon, Joshua Fineberg. Musique & mathématiques. Rencontres Musicales Pluridisciplinaires GRAME, Musiques en Scène1996, Aléas Editeur, Lyon 1996.

Rapports d'études

[34] Un reconocedor de notas musicales. Carlos Agon, Tesis de Ingenieria de Sistema y computacion, UNIANDES, Bogota, 1990.

[35] Quantification Rythmique. Augusto Agon. DEA d'informatique de l'université Paris XI. Paris, 1994.

[36] OpenMusic : Un langage visuel pour la composition musicale assistée par ordinateur. Augusto Agon, Thèse de doctorat de l'Université Paris 6, 1998.

Rapports techniques

[33] Chant pour PatchWork, Manuel d'utilisation. Ircam, 1992.

[37] PatchWork Script. Ircam, 1995.

[38] PatchWork Midi Manager. Ircam, 1995.

[39] PatchWork Rhythm Quantification Editor. Ircam, 1994.

[40] OpenMusic User manual. Ircam, 2003.

[41] OpenMusic Developer's Documentation. Ircam, 2000.

Annexe A The OM Composer's Book

Table des matières

By Francesco Filidei

[om11] Symétrie, géométrie et aléatoire
By Allain Gaussin

[om12] « Fondre » écriture et improvisation
By Jean-Rémy Guédon

[om13] Ikonological transfer: from Sound to Score or the Topological overview in “Das Werden im Vergehen”
By Karim Haddad

[om14] Encore (1999)
By Jean-Luc Hervé

[om15] Architecture as Virtual Music: re-actualizing *Zonnestraal*
By Christian Jaksjø

[om16] Navigation of Structured Material in "second horizon" for Piano and Orchestra (2002)
By Johannes Kretz

[om17] The genesis of "Aschenblume" and the role of Computer Aided Composition software in the formalisation of musical processes.
By Mauro Lanza

[om18] Modeles et transpositions dans « Voirex »
By Philippe Leroux and Frederic Voisin

[om19] Hérédo-Riobotes
By Fabien Levy

[om20] Os pour voix d’homme et électronique
By Paola Livorsi

[om21] METRO, ambiances et sons du métro parisien avec orchestre
By Claudy Malherbe

[om22] "Ecriture, Fractals et synthèse, contrôle par courbes"
by Mikhail Malt

[om23] "Génération de processus mélodiques, harmoniques et rythmiques pour l'opéra "K...""
by Philippe Manoury et Serge Lemouton

[om24] Exploitation des données extra-musicales dans le processus compositionnel.
By Alexander Mihalic

[om25] Notes on A Collection of Caprices
By Paul Nauert

[om26] Sculpted Implosions: some algorithms in a waterscape of musique concrète
By Ketty Nez

[om27] Synthèse sonore et mise en espace dans "Stelae for the failed times"
de Brian Ferneyhough.
By Gilbert Nouno

[om28] Klangspiegel
By Luis A. Pena

[om29] Contrôle de la synthèse granulaire avec OM-Csound
By Laurent Pottier

[om30] Epitaphe, pour grand ensemble et électronique d' Antoine Bonnet
By Camilo Rueda

[om31] "Kalejdoskop for clarinet, viola and piano."
By Örjan Sandred

[om32] Translating MusicV files to Csound format with the help of OpenMusic
By António Sousa Dias

[om33] Calculation of Proportional Segments in *Diastema*.
By Oliver Martin Schneller

[om34] "verblich" for double-bass and electronics: the sinus-sounds, an application of om-Chroma
By Stephanie Schweiger

[om35] SAKSTI (2001)
By Georgia Spiropoulos

[om36] How CAC influences écriture
By Kilian Sprotte

[om37] Temporal Pivots and Constraint-based Harmony in élet...fogytiglan
By Marco Stroppa

[om38] Derrière la Pensée - (2000/2001). To touch the inner sound, before it becomes music; to dream about dreams, before they become real
By Elaine Thomazi Freitas

[om39] Controlling sound treatment with OpenMusic
By Hans Tutschku