CAO : vers la partition potentielle

Gérard Assayag

Cao: vers la partition potentielle

Gérard Assayag

Les Cahiers de l'Ircam: ***La composition assistée par ordinateur*** **1(3), juin 1993**

Le terme générique d'informatique musicale masque aujourd'hui une impressionnante diversité d'activités, de disciplines et de techniques. Certaines sont fortement liées à des domaines scientifiques établis (comme le traitement de signal pour la synthèse et la transformation sonores), d'autres ont un contour moins clairement perçu par le public: c'est le cas de l'ensemble des démarches dénommées composition assistée par ordinateur (CAO). En effet, la composition est présente, à l'état de projet ou de réalisation, dans tous les secteurs de l'informatique musicale; par conséquent, pourquoi constituer en discipline autonome cette alliance de la technologie et de la composition? Nous proposons cet élément de réponse: *la spécificité de l'écriture musicale*, entendue d'abord comme écriture pour les instruments, et la nécessité pour une pensée musicale en devenir, en mutation perpétuelle, d'accéder, comme le font aujourd'hui tous les systèmes de pensée formalisée, à la puissance et la souplesse de l'outil informatique. L'écriture instrumentale semble alors un champ d'étude à la fois précis et ouvert: elle constitue un modèle, encore inégalé, d'adéquation entre des systèmes d'opérations combinatoires sur des ensembles de symboles et un univers sonore disposant de ses propres règles de fonctionnement perceptif et cognitif, la notation opérant un lien cohérent entre ces deux instances. Par là, elle ouvre aussi à l'intégration de matériaux sonores nouveaux - extensions du monde instrumental, synthèse et transformation sonores. Le développement d'une technologie informatique spécifique se justifie alors de plus en plus au fur et à mesure que s'étend son champ d'action et que se complexifient ses procédures. Nous emploierons donc le terme «aide à la composition» en privilégiant, parmi toutes les interprétations possibles, celle qui se rapproche le plus de l'idée d'écriture instrumentale dans ce qu'elle porte d'exemplaire quant à la formalisation, à la codification, à la notation et à la relation au monde physique. Les systèmes de CAO seront au moins capables d'apporter une aide efficace dans le cas spécifique de l'écriture pour instruments ; ils devront constituer une passerelle cohérente entre ces derniers et les nouveaux champs sonores. Ils pourront à terme constituer de bons modèles pour le contrôle de la synthèse pure.

Avant de dresser un bref historique des travaux de l'Ircam dans ce domaine, nous proposons de poser très schématiquement, et à la lumière de cette expérience, un cadre conceptuel pointant les caractéristiques requises par un système idéal d'aide à la composition. Il nous semble pertinent de classer ces dernières selon trois axes: les aspects de langage, ceux de notation et ceux de perception.

Langage

Tout d'abord, il faut mettre en adéquation deux activités d'un ordre symbolique et combinatoire: d'un côté, la recherche inlassable de nouveaux éléments de langage musical et de nouveaux modes de structuration; de l'autre, la spécification et l'exploitation interactive d'algorithmes permettant d'actualiser ces idées et d'en explorer systématiquement les conséquences techniques et artistiques.

Pour servir de support à cette «rencontre» intellectuelle, on choisit souvent Lisp, un langage de programmation issu des recherches en intelligence artificielle qui constitue probablement le meilleur compromis, pour sa simplicité et son homogénéité syntaxique, son interactivité, son expressivité (relation directe entre les expressions symboliques du langage et les structures qu'elles représentent). On réalise généralement une version spécialisée de ce langage qui permet au musicien de se concentrer sur la seule expression de problèmes musicaux. Des opérations prédéfinies pour les objets usuels de la musique et une syntaxe aménagée pour faciliter leur combinatoire sont alors proposées.

Notation

Construire des structures, exprimer des calculs n'a 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 le langage 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.

Perception

Les manipulations sur un matériau compositionnel ont un caractère éminemment combinatoire. Toute l'entreprise pourrait rester au stade de la spéculation formelle si un lien solide n'était pas établi avec la perception. Plusieurs principes peuvent être appliqués pour arriver à cette fin: la constitution de tests auditifs à partir des sorties musicales, la connexion du logiciel de CAO à des outils d'analyse et de synthèse, ou enfin l'intégration de modèles acoustiques et psychoacoustiques au sein même du logiciel. Ainsi, un environnement d'aide à l'écriture instrumentale permettra d'indiquer certains effets perceptifs saillants et, par là, de mieux intégrer des éléments hétérogènes (instruments, synthèse, objets sonores, contrôle spatial).

La composition assistée par ordinateur à l'Ircam

Les recherches menées de 1984 à 1986 par Claudy Malherbe et l'auteur au sein de la cellule de recherche «Instruments modèles écriture» [1] ont constitué la première tentative structurée d'intégration des caractéristiques de langage, de notation et de perception dans un outil général de CAO. L'environnement logiciel conçu était écrit en Lisp et fonctionnait sur un réseau de mini-ordinateurs Vax, les stations personnelles de type Macintosh étant peu répandues. Un sous-langage spécialisé permettait de définir des structures musicales sous formes d'expressions algébriques qui étaient soumises à un interprète de calcul formel (similaire à ceux utilisés par exemple en résolution d'équations ou en intégration) ; des bibliothèques de formules modélisant des structures musicales (systèmes harmoniques, modes de hauteurs, structures rythmiques, etc.) pouvaient être chargées. Les résultats, apparaissant sous forme de structures polyphoniques complexes, étaient finalement traduits en notations musicales et imprimés. Un important travail de constitution de bases de données d'analyses acoustiques et psychoacoustiques d'objets sonores (notamment instrumentaux) a été parallèlement mis en chantier. Une connexion était proposée au sein du logiciel de CAO sous la forme d'un algorithme très général de constitution de graphe harmonique exprimant, pour une base de données choisie, l'ensemble des relations exploitables dans la perspective de l'écriture d'une séquence instrumentale intégrant ces objets (il était alors possible de mêler dans l'orchestration sons instrumentaux, concrets ou de synthèse, tout en gardant un contrôle fin de la structure harmonique).

Jusqu'en 1990, l'environnement preForm/Esquisse, écrit principalement par Lee Boynton, Jaques Duthen et Magnus Lindberg, a associé, sur le Macintosh d'Apple, de nombreuses librairies de règles compositionnelles [2] destinées à la génération et à la manipulation de matériaux, des éditeurs graphiques interactifs, la communication avec les instruments Midi et une utilisation intensive de la technologie objet (un modèle informatique dans lequel des objets logiciels disposent d'une compétence locale et échangent des messages). Mentionnons aussi le programme Carla de Francis Courtot (1991) qui intègre la notion de représentation logique et d'apprentissage.

Plus récemment, le programme PatchWork [3] a introduit la notion de programmation visuelle (le compositeur construit des patches, diagrammes fonctionnels dans lesquel sont articulés des modules représentant des opérateurs musicaux). Ecrit en Lisp, PatchWork dispose d'éditeurs graphiques très puissants et capitalise une partie des connaissances musicales et technologiques issues des environnements qui l'ont précédé [4] .

Situations, un projet mené actuellement par Camilo Rueda et Antoine Bonnet, propose une vision alternative dans laquelle le compositeur formule un système de contraintes (par exemple des contraintes harmoniques sur les intervalles constituant des accords) et laisse le programme construire un objet musical qui constitue une solution de ce système. Très puissant dans son cadre d'expérimentation actuel (principalement harmonique), ce logiciel prometteur devra être étendu pour embrasser des problématiques musicales plus variées.

Si le progrès s'est manifesté dans le sens d'une expressivité accrue des langages informatiques et dans l'apparition de véritables éditeurs graphiques, la représentation cohérente de la trilogie qui guide notre propos - langage, notation et perception - reste un idéal à atteindre. En particulier, le niveau de la notation devra tendre à la même modularité que celui de la programmation, de façon à constituer à terme un système d'interface programmable et personnalisable intégrant contraintes et connaissances musicales. Pour cela, les concepts originaux développés dans le domaine de la programmation visuelle devront trouver leur contrepartie dans celui de la notation. Ainsi, le compositeur programmera-t-il lui-même, en toute liberté, le calcul fonctionnel qui déterminera les contours de son champ d'expérimentation musicale, les contraintes qui informeront ce dernier et lui imprimeront une structure, les degrés de liberté qui permettront une navigation dans l'espace de connaissances alors constitué, les modalités sensibles de cette exploration : graphisme, son, interaction gestuelle. Enfin, il pourra choisir de masquer toute l'étape d'élaboration en définissant une *partition potentielle* où se superposeront affichage des résultats, contrôle interactif des degrés de liberté, entrée des paramètres musicaux - sorte de feuille blanche «informée» par une hiérarchie d'obligations menant des contraintes de cohérence élémentaire du langage musical à celles, de plus haut niveau, exprimant les caractéristiques stylistiques personnelles.

Modèles informatiques

Au centre de cette idée de partition potentielle se trouve la notion de modèle informatique d'une structure musicale, plus puissante et plus générale que celle d'esquisse ou de processus employées jusqu'alors. Ce concept est au coeur des recherches actuelles dans le domaine de l'analyse comme dans celui de la composition assistées [5] : il n'est pas inutile de s'y arrêter quelque peu. Riotte et Mesnage [6] notent que, depuis une trentaine d'années, la légitimité de la simulation compositionnelle s'est établie comme démarche d'analyse assistée par ordinateur ; tout d'abord simulation stylistique [7] , puis reconstitution, à partir d'un modèle informatique, d'une partition complète [8] .

Mais qu'entend-on exactement ici par modèle ? De manière générale, ce dernier 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. Ainsi des modèles utilisés en synthèse des sons, tels la modulation de fréquence ou les modèles physiques [9] . Si la simulation stylistique entre bien dans cette définition, puisqu'elle est à même de produire un nombre indéfini d'instances musicales obéissant aux lois d'un genre, les modèles de partitions de Mesnage et Riotte constituent un cas extrême, dans la mesure où ils sont entièrement destinés à la restitution d'un objet unique. Ils ne constituent pas cependant une simple description de ce dernier - ;qu'ils généralisent dans la mesure où ils lui substituent une collection de mécanismes formels dont une paramétrisation particulière fournira l'objet final. Il est alors loisible d'envisager, en essayant d'autres jeux de paramètres, des *variantes* [10] ou encore, dans le cas d'espaces paramXSétriques porteurs de fortes discontinuités, des objets très éloignés de la référence mais partageant avec elle quelque communauté secrète de forme [11] . Ainsi, des lois d'un genre (et de la classe des modèles idoines) à la formalisation d'une oeuvre, il y aurait une transformation opérant dans le champ même des modèles, une *résolution* au

sens que donne Chemillier à ce mot. Pour cet auteur, en effet, une *résolution*, au sens mathématique, permet de passer «d'une définition équationnelle (équivalente à une liste de contraintes) de l'ensemble des séquences musicales recherchées à une définition paramétrique équivalente» [12] .

Par ce tour de passe-passe consistant à jouer sur l'ambiguïté du terme modèle (le modèle est à la source de l'oeuvre, l'oeuvre constitue le modèle destiné à être pastiché lors de la simulation), l'analyse moderne tend à se situer dans la catégorie de la création. En effet, la démarche de l'analyste utilisant des outils scientifiques n'est plus très différente de celle du compositeur confronté aux mêmes outils. Cela reste vrai jusque dans la distorsion du concept de modèle, que chacun opère pour son compte: dans la finalité qu'ils lui forgent de ne devoir engendrer qu'une singularité - singularité passée dans le cas de l'analyste, singularité à venir dans le cas du compositeur [13] .

C'est dans leur action sur le modèle que divergent cependant les deux démarches. Dans le cas analytique, les raffinages successifs du modèle ont tendance à minimiser le nombre de paramètres - le cas idéal étant leur disparition complète, les valeurs des paramètres d'un niveau de formalisation se trouvant engendrées par un formalisme de niveau supérieur. Ainsi, sauf dans le cas des variantes, ce n'est pas tant l'expérimentation sur le modèle - la simulation - qui prévaut que son raffinement progressif à partir du stade de la paraphrase jusqu'à celui de l'explication.

Pour le compositeur, au contraire, seule l'expérimentation productive compte. Le modèle est alors manipulé selon deux modalités: d'une part, les objets produits constituent des éléments de matériau structuré qui pourront être mis de côté au fur et à mesure de l'exploration. D'autre part, l'observation de ces éléments peut conduire à la remise en question de la théorie musicale sous-jacente, qui sera remaniée en conséquence, conduisant à l'élaboration d'un nouveau modèle. On retrouve alors le concept classique de simulation comme validation de la théorie, à ceci près que - et c'est une différence fondamentale avec la démarche scientifique - le phénomène de référence permettant de conduire la comparaison validante n'existe pas autrement qu'à l'état d'idéal dans l'imaginaire du compositeur. Aux fonctions répertoriées du modèle scientifique, le compositeur aux prises avec la machine à simuler qu'est l'ordinateur ajoute une perspective téléonomique: on retrouve la polarisation du procédé vers une singularité, en un étrange mixte, inimaginable avant l'avènement de la technologie informatique, où se côtoient une méthodologie scientifique à tendance généralisante et la notion irréductible d'oeuvre unique.

Le modèle informatique est aujourd'hui à la jonction de l'analyse et de la composition. La discussion n'a cependant porté jusqu'ici que sur la notion de modèle en général. Qu'en est-il de la spécificité informatique ? Elle est tout d'abord pragmatique. Il est banal mais important de rappeler que le tout-numérique donne sa pleine puissance à la notion de simulation (encore qu'il y ait eu des exemples de simulation compositionnelle analogique). Plus significativement, il existe des correspondances fortes entre les problématiques rencontrées dans les diverses branches de la recherche musicale informatique : par exemple, la conduite de processus parallèles obéissant à des contraintes verticales de synchronisation et de cohérence [14] implique, dans les deux cas, la maîtrise de formalismes réglant les relations temporelles des objets traités ; la même dialectique du hors-temps et du en-temps [15] est à l'oeuvre dans les deux disciplines et l'on y cherche des langages formels permettant de décrire des objets du point de vue tant de leurs propriétés invariantes dans le temps que de celles qui régissent la chronologie de leur existence [16] .

Enfin, l'informatique a tendance à unifier les deux instances, explicative et générative, du modèle. En effet, un programme d'ordinateur est d'abord un *texte*, exprimant avec les conventions d'un langage formel un réseau de relations. Il donne lieu, lors de son exécution, au déploiement dans le temps d'un processus dont la forme est réglée par ces relations. Il reproduit ainsi lui-même, par une sorte de mise en abyme, la dialectique modèle/simulation. Le programme d'ordinateur, implémentant un calcul compositionnel ou une formule d'analyse, occupe donc une position idéale pour peu qu'il veuille bien exhiber sous une forme assimilable sa composante logique. C'est le pas, important à nos yeux, qui a été franchi avec la *programmation visuelle*, technique par laquelle on substitue au langage formel évoqué plus haut une représentation graphique schématique équivalente. Une des réalisations les plus avancées dans ce domaine est sans conteste le logiciel PatchWork.

Nous donnons plus loin un exemple détaillé en PatchWork qui montre comment la logique sous-jacente aux modèles de programmation peut traduire directement un problème musical.

Structure formelle et interprétation

Pour la logique formelle, un modèle est l'interprétation que l'on donne à un système formel dans un domaine particulier, en associant à chaque constante symbolique une valeur particulière du domaine, et à chaque prédicat sur **n** variables une application des n-uplés du domaine vers les deux valeurs de vérité [17] *vrai* et *faux*. Des tentatives ont été faites d'appliquer ce concept en linguistique, le modèle étant alors la sémantique de la langue naturelle [18] .

Nous nous inspirons de ce concept pour définir la démarche qui nous semble aujourd'hui la plus riche en informatique musicale: aider le compositeur en lui fournissant un système qui actualise élégamment la traversée du système formel - l'ensemble des calculs préliminaires servant de charpente à la composition - vers la structure musicale - interprétation concrète des dispositions formelles issues du calcul. Ces dernières ainsi plongées dans une *sémantique* - un contexte où elles prennent une signification musicale -, il devient possible de mesurer leur rentabilité, leur efficacité à produire des éléments de matériaux organisés, que le compositeur jugera éventuellement recevables selon son propre système de références.

Représenter sous une forme musicale une structure ou un processus abstraits - que l'on souhaite au départ aussi peu contraints que possible pour préserver la liberté d'imagination formelle - implique la maîtrise d'un système de représentations musicales, c'est-à-dire d'un ensemble cohérent de structures symboliques aptes à définir les propriétés des objets musicaux et les relations que ces derniers entretiennent à mesure qu'ils se combinent pour former des assemblages de plus en plus complexes. Le système de représentation devra rendre compte de tous les niveaux d'intégration, depuis les ensembles de points et d'intervalles dans les domaines paramétriques de base (hauteurs, durées, intensités, etc.) jusqu'aux systèmes d'agrégation verticale et horizontale (harmonie, polyphonie). La difficulté réside alors dans le repérage et l'extraction d'universaux qu'il sera pertinent de fixer en des représentations informatiques de référence. Une telle étude, fondamentale dans les deux sens du terme, est trop souvent négligée dans la conception des logiciels de CAO, cela conduisant à un déficit de généralité des modèles et donc à des goulots esthétiques qui provoquent de fortes réactions de rejet de la part d'utilisateurs potentiels. Pourtant, seule la maîtrise de ce niveau profond serait en mesure de garantir et l'ouverture stylistique - la capacité à satisfaire des compositeurs adoptant des points de vue très éloignés les uns des autres - et une communication mieux contrôlée entre pensée musicale et systèmes de production sonore (notamment par synthèse).

La description formalisée des structures et des opérateurs régissant des ensembles dénombrables d'objets ressort, en informatique, de la problématique des *types* [19] de données. La méthodologie des langages-objets [20] , très utilisée aujourd'hui, n'offre qu'un cadre, certes pratique, mais n'atteignant pas aisément à la spécificité des types d'objets musicaux. Une approche possible consiste à opérer une formalisation purement algébrique [21] , puis à traduire celle-ci au mieux en fonction des outils de typage informatiques disponibles. Une autre approche, explorée actuellement à l'Ircam par Camilo Rueda, consiste à considérer un type comme un domaine de valeurs pour une variable musicale (par exemple, un accord de quatre notes au sein d'un réseau harmonique), ce domaine se déduisant de domaines plus généraux (l'ensemble des accords de quatre notes) par une série de *contraintes* qui en restreignent l'étendue (par exemple, des relations d'intervalles locales - internes à l'accord - ou translocales - mettant l'accord en relation avec son contexte horizontal ou vertical) [22] . Un mécanisme général est alors fourni qui engendre l'ensemble des objets candidats (par exemple, des séquences harmoniques) une fois définies les contraintes, sans que l'utilisateur n'ait à se préocupper des modalités de cette génération. L'avantage de ce point de vue réside dans le fait que les relations de dépendances liant les objets musicaux et traversant les frontières des niveaux d'organisation sont pensées dès la conception des types musicaux de base. Ainsi, l'objet élémentaire ne peut-il pas être conçu hors du réseau contextuel de relations dans lequel il est pris à toutes les étapes de l'élaboration du matériau. Il suffirait alors de considérer le cas particulier d'un contexte vide pour rendre compatible la typologie par contraintes avec l'approche algorithmique plus traditionnelle dans

laquelle les objets sont construits selon un schéma fonctionnel : un résultat unique est obtenu par la description explicite des mécanismes de sa construction. Ce schéma préside à l'utilisation du logiciel PatchWork (voir exemple 1), qui sert-lui même d'environnement de prototypage au moteur de résolution de contraintes.

Les deux logiciels de CAO présentés en exemple, PatchWork (exemple 1) et Boards (exemple 2 et exemple 3), montrent deux approches possibles de la relation calcul/interprétation. Dans Boards, l'utilisateur définit ses calculs directement au niveau du langage sous-jacent (Lisp). Il dispose en revanche d'une puissante mécanique de génération d'interfaces graphiques pour construire non seulement l'interprétation en termes musicaux mais aussi la navigation au sein des structures engendrées. Dénué de cette logique d'interface utilisateur, PatchWork intercale en revanche une couche de programmation visuelle [23] entre le Lisp et l'utilisateur, ainsi que des mécanismes de typage qui contraignent les connexions possibles entre modules. Les types (notes, accords, séquences d'accords, structures métriques/rythmiques, etc.) n'obéissent malheureusement pas à une organisation cohérente (ils sont largement indépendants les uns des autres), ce qui se manifeste par une «mauvaise circulation» des objets [24] dans le circuit fonctionnel. La grande originalité de ce logiciel réside dans l'existence de modules éditeurs qui peuvent être vus indifféremment comme des boîtes fonctionnelles (traitant des entrées et élaborant des sorties) ou comme des éditeurs graphiques permettant la visualisation et la transformation manuelle du dernier état calculé à ce noeud du circuit. Par ce biais, calcul et interprétation sont très finement intégrés, un patch visuel servant finalement de *modèle* aux deux sens évoqués tout au long de cet article : représentation locale schématisée du processus algorithmique mis en oeuvre d'une part, interprétation dans le champ musical du formalisme associé d'autre part.

Exemple 1

- *Analyse de la trame (piccolo, xylophone, célesta, 2e violon) des mesures 14 à 30 dans Melodien pour orchestre de G. Ligeti (1971).*
- *Formalisation par Marc Chemillier dans le cadre de sa collaboration avec l'équipe Représentations musicales de l'Ircam. Réalisation en Patchwork par M. Chemillier, avec le concours de l'auteur.*

Dépassant la forme de surface qui suggérerait un développement mélodique par augmentations progressives à partir de petites cellules initiales, Chemillier a démontré le mécanisme sous-jacent qui résulte plutôt, pour chaque voix, de la conduite de deux processus itératifs opérant indépendamment l'un de l'autre sur un groupe de dix notes qu'ils répètent en transformant progressivement. Le contenu de hauteurs du groupe à un moment donné est appelé un état du processus. Les *états* successifs sont accumulés en une séquence et associés à un mécanisme rythmique simple pour former une voix. A chaque itération, le premier processus opère, à des positions choisies du groupe courant, des transpositions vers le bas («affaissements»), par des intervalles allant de la seconde mineure à la tierce mineure. Le second processus («effacements») oblitère à chaque itération des positions choisies du groupe courant (les groupes contiennent toujours dix positions, un effacement provoquant simplement le marquage d'une position comme vide). Les deux automatismes sont indépendants dans le sens où les états affaissés se succèdent sans tenir compte des effacements, qui vont se produire sur des copies de ces états, engendrant un flux parallèle. De cette manière, les affaissements continuent de se produire de façon souterraine, sur des positions qui se trouvent oblitérées pendant un temps puis qui réapparaissent, transposées vers le bas.

On construit la séquence finale en distribuant les hauteurs sur une pulsation rythmique régulière (par une division des temps en 5, 6, 7 et 8 unités selon la voix, les impulsions étant regroupées deux par deux), en intercalant une unité de silence aux frontières des groupes (états) de hauteurs et en ignorant purement et simplement les positions effacées. Les groupes mélodiques ainsi segmentés sont de longueurs variables, masquant au niveau de la surface musicale la notion de processus opérant sur une série de positions de longueurs constantes.

La figure 1 montre le *patch* (diagramme fonctionnel d'opérateurs et de données) construisant la trame.

Chaque module vertical construit une voix. Les résultats issus des quatre modules sont collectés au niveau de la boîte nommée `trame`. Des sous-ensembles de la partition n'appartenant pas à la trame modélisée ont été entrés manuellement dans les boîtes `tenues` et `mélodies` de manière à donner une interprétation plus réaliste.

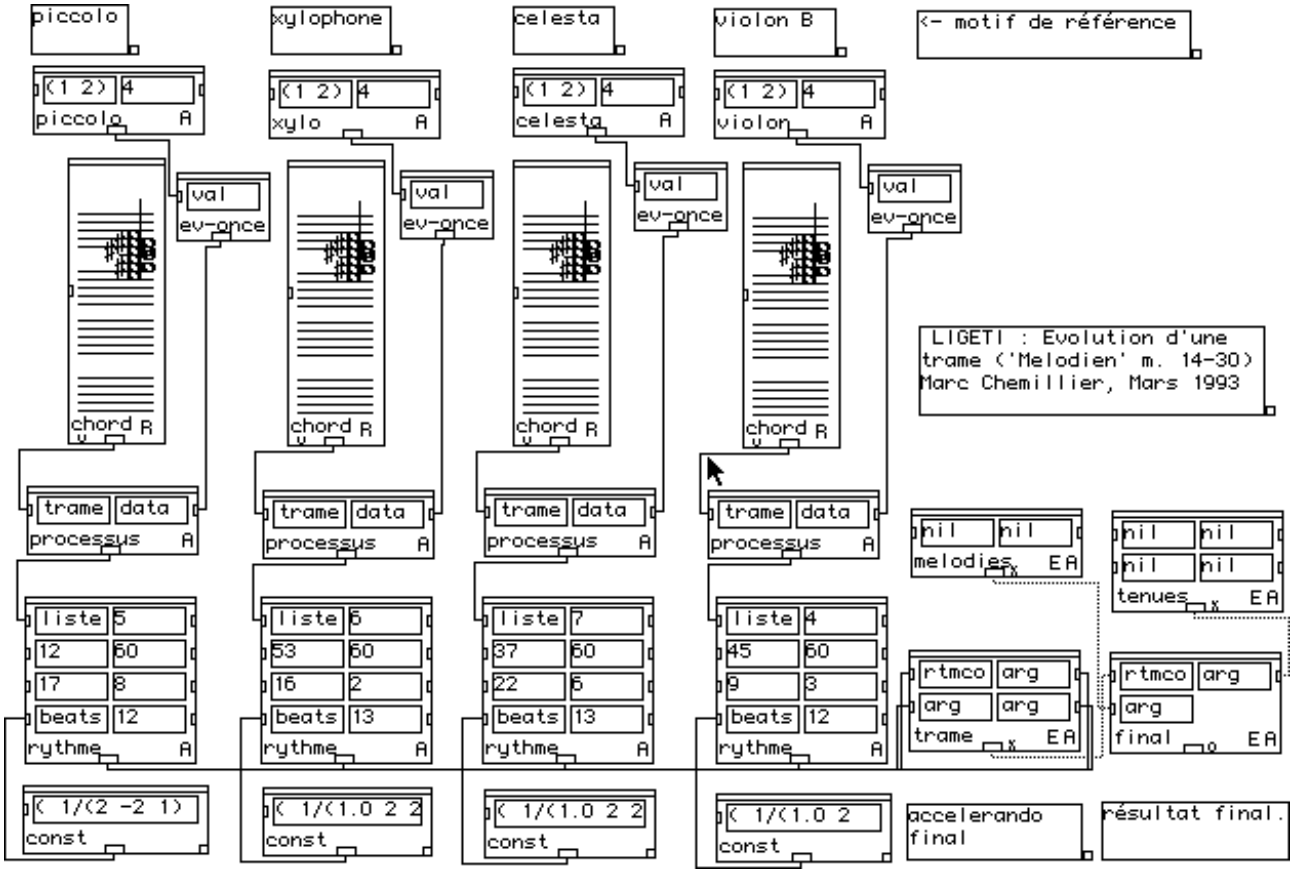

Figure 1

Dans chaque voix, la boîte d'accords contient une séquence de dix notes qui constitue l'état de référence. Celui-ci est montré pour le piccolo figure 2 , les trois autres en sont dérivés par simple permutation. L'abstraction (boîte masquant un réseau plus complexe) nommée `processus` contient la définition des opérateurs d'affaissement et d'effacement. La boîte `rythme` est responsable de la mise en temps des séquences de hauteurs selon le mécanisme décrit plus haut.

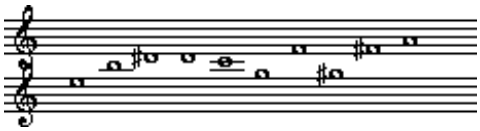

Figure 2.

En ouvrant l'abstraction `processus`, on accède au sous-patch d'affaissement montré à la figure 3 . La boîte `pwreduce` est un itérateur qui parcourt une liste d'objets et applique un opérateur binaire à l'objet courant, ainsi qu'au résultat de la précédente application. Il a donc une action cumulative qui est exploitée ici pour représenter notre processus. La liste à parcourir contient les paramètres du traitement sous la forme de sous-listes de positions à affaisser pour chaque état. L'accumulateur est formé d'une liste singleton contenant la série de dix notes de référence. L'opérateur prend le premier élément de l'accumulateur (c'est-à-dire le dernier état calculé), l'affaisse selon la liste de positions courantes et l'agrège par la gauche (opérateur `cons` en Lisp) à l'accumulateur. Ce dernier finit donc par contenir la séquence des états.

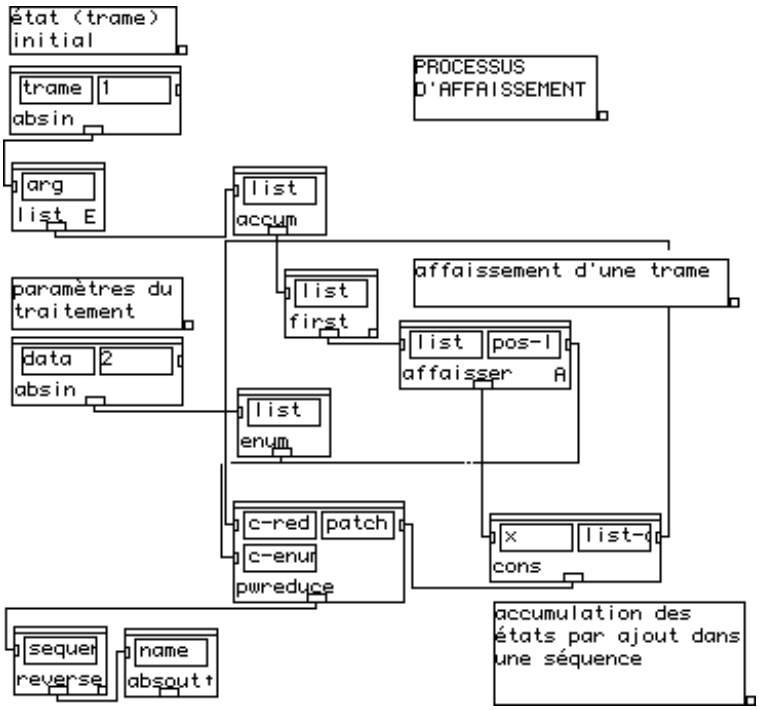

Figure 3.

Il suffit de remplacer la boîte `affaisser` par une abstraction quelconque implémentant un algorithme de complexité arbitraire pour simuler un autre processus. La nouvelle abstraction devra alors constituer une fonction de transition d'état. Ce mécanisme très simple est intéressant par sa *généricité*: toute transformation peut être utilisée comme fonction de transition, n'importe quelle définition peut être donnée de ce qu'est un état. Notamment, des états beaucoup plus abstraits que ceux utilisés dans cet exemple peuvent être associés à des éléments de la structure profonde [25] ou à des collections de paramètres pour la synthèse [26] .

Notons enfin que ce patch devrait devenir une primitive du système, puisqu'il correspond à un mécanisme simple et universel. On mesure ici comment une typologie musicale se construit en parallèle avec une typologie algorithmique assortie d'une représentation intuitive - ici graphique - dans un système d'aide à la composition au fur et à mesure que des problèmes d'analyse et de composition sont rencontrés et résolus.

La figure 4 donne la trame des mesures 9 et 10. Les états exprimés pour le piccolo sont les 13e, 14e et 15e. Voici les paramètres du piccolo:

```
;;; paramètres pour effacements
;;; no de l'état et liste de positions à effacer
(...
(13 (0))
(14 (6))
(15 (4))
...)
;;; affaissements par demi-tons
;;; no de l'état et liste de positions à affaisser
( (14 (1 2 4 8))
 (15 (2 8))
...)
;;; affaissements par tons
;;; no de l'état et liste de positions à affaisser
( (13 (3))
 (15 (0 5 6 7))
...)
```

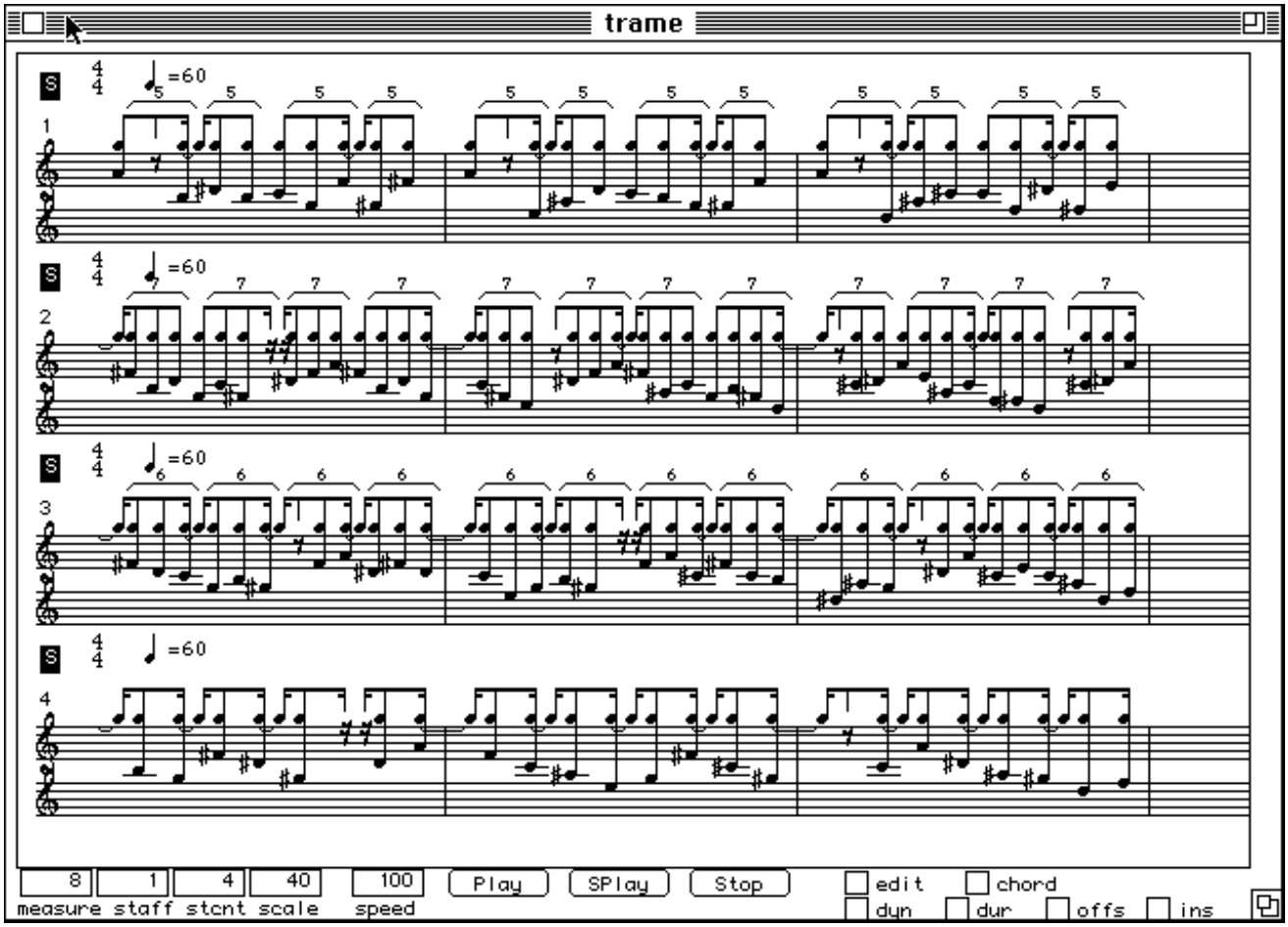

Figure 4.

Exemple 2

- *Génération : une application dans l'environnement Boards.*
- *Formalisme de composition par P. Haïm et A. Champert.*
- *Programme Génération par G. Assayag et F. Guédy.*

Soit deux éléments de l'ensemble S des couples (fréquence, durée) `s1 = (f1, d1)` et `s2 = (f2, d2)`. On examine la transformation `[[partialdiff]]` qui à s1 et s2 associe `s3 =[[partialdiff]](s1,s2) =(f3, d3)` défini de la façon suivante:

```
si d1 <= d2 alors f3 = (2f1+f2) / 2 et d3 = (2d1+d2) / 2
si d1 > d2 alors f3 = (2f1+f2) / 4 et d3 = (d1+2d2) / 2
```

En partant d'un noyau (s1, s2), et en utilisant [[partialdiff]] comme un itérateur sur l'ensemble des parties de s, on développe le mécanisme de génération suivant :

```
s1 s2
-----
s1 s2
s3 = [[partialdiff]](s1,s2)
-----------
s1 s2 s3
s4 = [[partialdiff]](s1,s3)
s5 = [[partialdiff]](s2,s3)
-----------
s1 s2 s3 s4 s5
s6 = [[partialdiff]](s1,s4)
s7 = [[partialdiff]](s1,s5)
s8 = [[partialdiff]](s2,s4)
```

```
s9 = [[partialdiff]](s2,s5)
s10 = [[partialdiff]](s3,s4)
s11 = [[partialdiff]](s3,s5)
------------

.......
```

Chaque étage, obtenu par ajout des recombinaisons exhaustives des objets de l'étage précédent, est appelé un *substrat*. Il existe des relations critiques entre les valeurs initiales s1 et s2 conditionnant l'existence de *régions* caractéristiques dans `S x S`. Par exemple, les relations `d1<d2/2 et f1<f2/2` définissent la région R1 telle qu'un substrat initié à partir de R1 s'écrira, en exprimant tous les éléments par rapport au noyau :

```
(1)

s3 : f3 = (2f1+f2) / 2 d3 = (2d1+d2) / 2
s4 : f4 = (6f1+f2) / 4 d4 = (6d1+d2) / 4
s5 : f5 = f1+f2 d5 = (2d1+d2)
........
```

Chaque région définit de la sorte une évolution type (une «enveloppe» fréquence/durée) à partir d'un couple de départ. Autour des valeurs critiques (frontières de régions), il existe une forte *dépendance aux conditions initiales* : des couples très proches initialement peuvent rapidement être canalisés vers des enveloppes différentes.

Dans un substrat de région donnée, on peut réécrire les égalités de manière à faire apparaître des structures interprétables en termes acoustiques. Ainsi, en posant:

```
f2 = x f1
```

les relations (1) pourront se réécrire :

```
f3 = 2(f1/2) + x(f1/2)
f4 = 6(f1/4) + x(f1/4)
...
f7 = 3(f1/2) + x(f1/2)
f8 = 6(f1/4) + 3x(f1/4)
...
f10 = 4(f1/2) + x(f1/2)
```

On peut voir que f1/2 constitue la fondamentale f0 d'un spectre décalé fn = nf0 + Kf0. f3, f7et f10 constituent les partiels de rang 2, 3 et 4 de ce spectre. Le facteur K exprime la proportion de la fondamentale constituant le décalage.

Ainsi, chaque substrat peut être divisé lui-même en plusieurs sous-ensembles de partiels d'une fondamentale f0, avec un facteur de décalage K constant. On étudiera alors les groupes de fondamentales différentes à K constant ou les groupes à f0 constant et K variant en utilisant x comme variable de contrôle. La connexion de ces groupes entre eux devrait, selon les auteurs, permettre une mise en rapport contrôlée du taux d'inharmonicité global obtenu avec les taux locaux associés à chaque région du substrat [27] .

La taille des substrats croît de façon exponentielle, rendant leur exploration malaisée. Le choix qui a été fait pour l'implémentation de Génération dans l'environnement Boards est fondé sur la constatation suivante : un substrat engendré à partir d'un noyau choisi dans un autre substrat se

trouve inclus dans le premier. Ainsi, l'on peut construire une exploration partielle en relayant des dérivations successives à partir de noyaux choisis dans le dernier sous-substrat calculé. L'intérêt de la méthode est qu'elle permet d'atteindre une profondeur arbitraire (voir figure 7).

La figure 5 montre un écran de contrôle du programme Génération. La zone 1 est la partie générique qui est héritée par toute application définie comme une sous-classe de l'application score dans Boards. Cette

notion sera explicitée plus loin. La zone 2 permet de spécifier un noyau (s1, s2) par la donnée des fréquences et des durées, ainsi que de déclencher le calcul d'un substrat une fois précisé l'âge maximal de Génération. Le substrat courant apparaît dans la zone 5 sous la forme d'une liste de couples (fi, di). On peut relancer une dérivation à partir de cette liste en y sélectionnant deux couples qui apparaissent alors automatiquement dans la zone 2, puis en déclenchant le calcul.

A chaque fois qu'un substrat est calculé, son noyau est ajouté à la liste de la zone 4. Il peut donc être rappelé en sélectionnant un item dans cette liste.

Dans la zone 7, les éléments du substrat courant sont interprétés en numéros de partiels avec décalage K, la zone 6 listant les fondamentales disponibles à un stade d'utilisation donné.

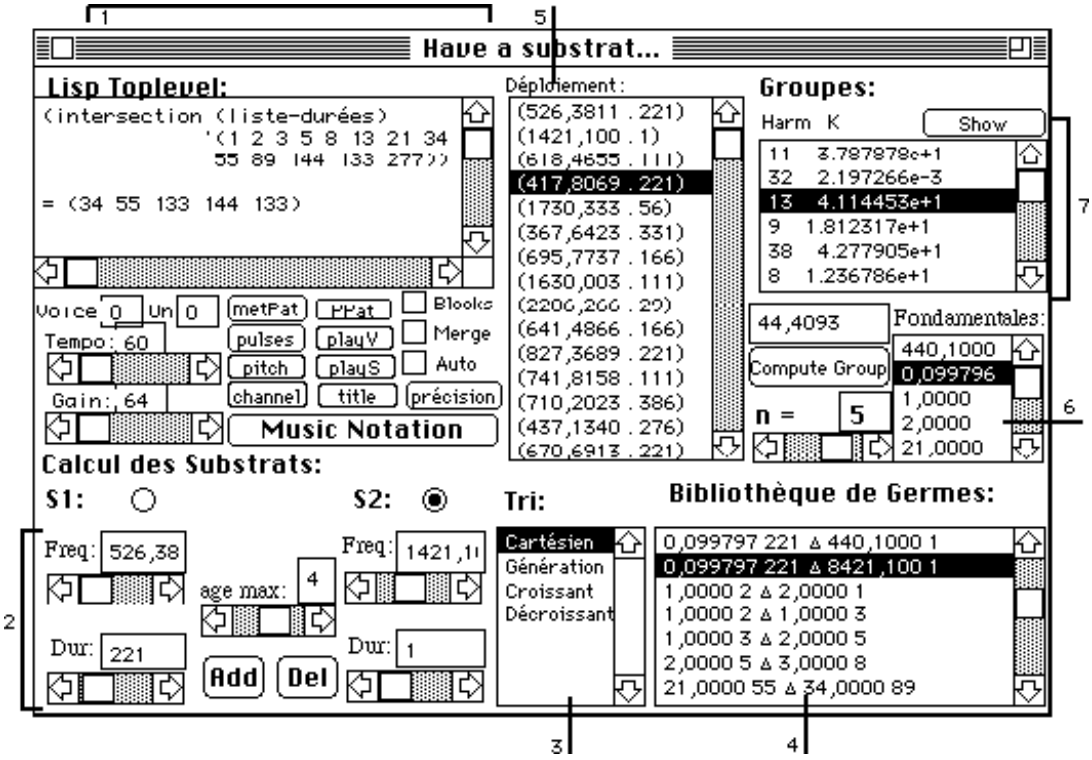

Figure 5.

Le substrats calculés apparaissent systématiquement dans la fenêtre *score* (figure 6), après un tri (non explicité dans cet article) et un choix de résolution temporelle qui fera éventuellement apparaître des événements très proches comme des accords.

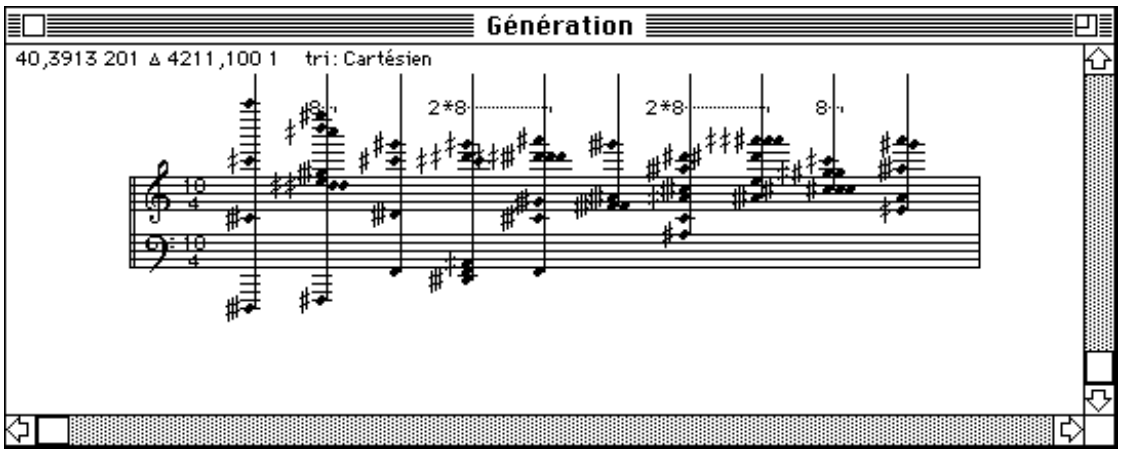

Figure 6.

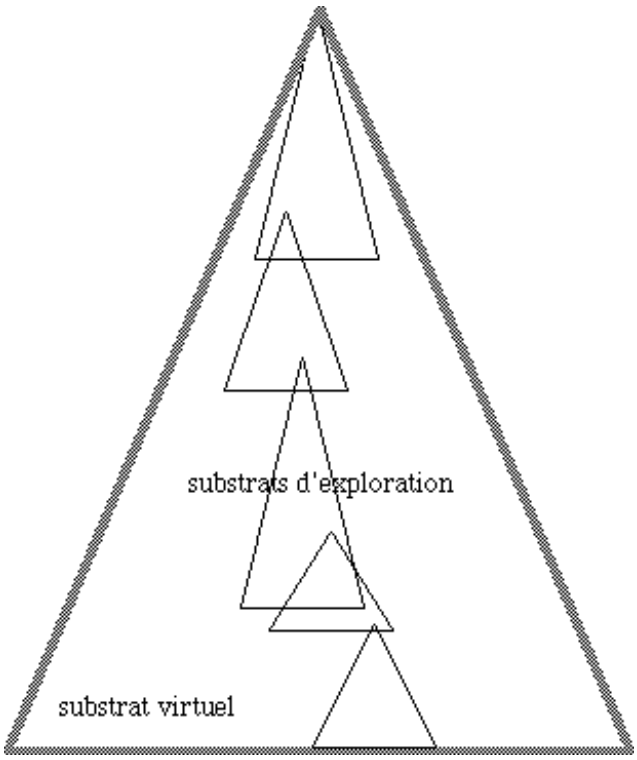

Figure 7.

Exemple 3

- *Boards : logiciel d'aide à la composition réalisé par G. Assayag, en 1987-1988.*
- *Boards a été employé notamment par : Claudy Malherbe, Klang (1990) pour vents et percussions, PlayBack (1991) pour soprano, ensemble instrumental et électronique, Masques (1991) pour sept instruments ; Philippe Haïm, Timeless, pour treize musiciens (1992).*
- *Il a été utilisé dans un contexte pédagogique à l'université de Caen.*

Ce logiciel a été conçu en vue de faciliter la réalisation d'applications musicales dans un environnement Lisp [28] sur Macintosh. Offrant un certain nombre d'extensions à ce langage, notamment une *orientation objet* [29] articulée de façon cohérente avec le graphisme, il comporte des classes prédéfinies permettant de manipuler les éléments de l'interface Macintosh (menus, contrôles, fenêtres, etc.). Des classes de plus haut niveau autorisent la création instantanée d'éditeurs interactifs pour le texte, la programmation Lisp, les fonctions mathématiques ou la musique. Des classes, prédéfinies elles aussi, définissent la notion d'application multidocument et multifenêtre. Boards offre, enfin, un générateur d'interfaces homme-machines.

Un *board* est une fenêtre conçue comme un panneau de contrôle, dotée d'un ensemble de commandes (boutons, barres de défilement, etc.) et de zones d'édition textuelle ou graphique servant comme autant de *vues* cohérentes d'un même objet - texte, structure musicale, etc. - qui pourra être ainsi édité, exploré, archivé, transformé.

Une *application* est un objet auquel sont associés des fenêtres de contrôle (boards) et des menus. Les commandes issues de ces derniers forment des messages envoyés à l'application ; de même, les fenêtres peuvent communiquer entre elles et avec l'application par l'envoi de messages. Une application regroupe ainsi un ensemble de fonctionnalités circonscrivant une activité particulière, comme l'édition de textes ou l'expérimentation sur les structures musicales (cf. figure 8).

La conception objet de cet environnement permet de définir un board à partir d'un autre plus général, tout en héritant des propriétés du premier ; de même pour les applications. Ainsi l'application *GasToplevel* (un

«écouteur [30] » Lisp) est-elle une fille de l'application *TextEdit* (un petit éditeur de texte), à laquelle il a suffi de rajouter la possibilité d'évaluer la zone de texte sélectionnée en appuyant sur une touche du clavier ; toutes les fonctionnalités d'édition de texte sont héritées par la nouvelle application dont l'écriture est ainsi considérablement simplifiée. L'application *ScoreBoard,* qui nous intéresse plus particulièrement ici, provient elle-même des précédentes. *ScoreBoard* contient un tableau de contrôle pour la construction, la manipulation, la visualisation et l'écoute d'objets musicaux. Permettant l'édition de documents multiples, la visualisation en notation musicale, l'impression des partitions, le contrôle Midi, elle intègre notamment un Toplevel Lisp (ce qui explique sa filiation avec TextEdit et GasToplevel) et des contrôles spécialisés pour l'affectation d'une sémantique musicale aux expressions Lisp.

Enfin, l'application *Board Editor*, un générateur d'interfaces, se trouve au coeur du logiciel puisqu'elle permet la spécification (graphique et interactive) de nouveaux boards et de nouvelles applications sans avoir à écrire de programme Lisp [31] .

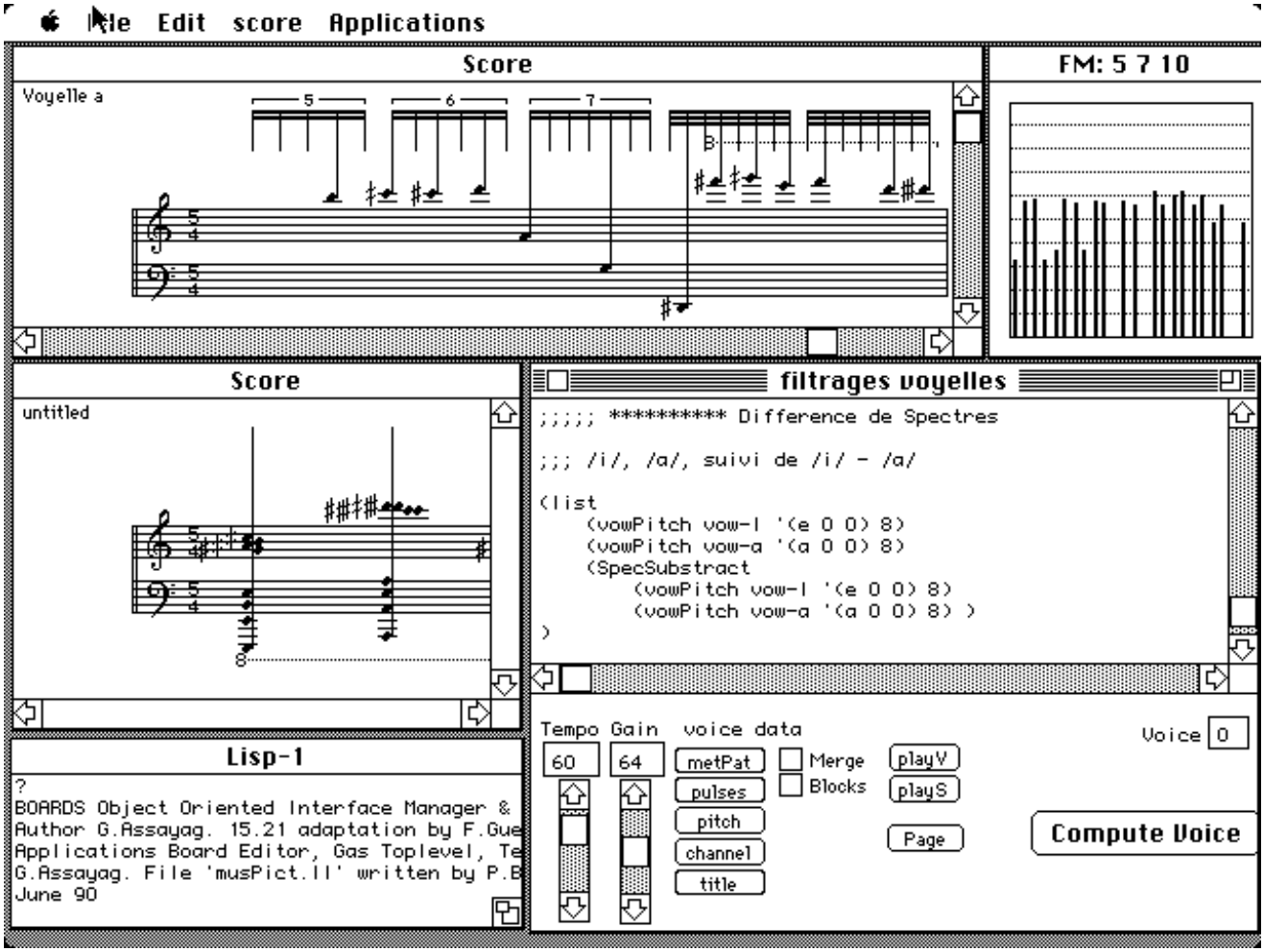

Figure 8.

L'application Board Editor

L'application Board Editor permet de définir de nouvelles applications sans écrire de code par la description de tableaux de bord contenant des petits objets graphiques (des *contrôles* dans la terminologie informatique) dotés d'un comportement par défaut. Des *messages* standardisés sont envoyés par ces objets à l'application à laquelle ils appartiennent à chaque fois qu'ils sont stimulés par une action de l'utilisateur. Les *méthodes* (actions associées à ces messages) peuvent être redéfinies en Lisp par l'utilisateur, qui confère ainsi des sémantiques particulières aux contrôles. La création des contrôles, leur configuration spatiale (position et de dimension), leur association à une sémantique sont complètement effectuées par le biais d'interface écran/souris du Macintosh. La figure 9 donne une idée du travail d'édition d'interface dansBoard Editor. Les mécanismes d'héritage inhérent aux langages objets permettent, en outre, au niveau

des contrôles, comme à ceux des boards ou des applications, de réutiliser certaines entités pour en créer de nouvelles par spécialisation.

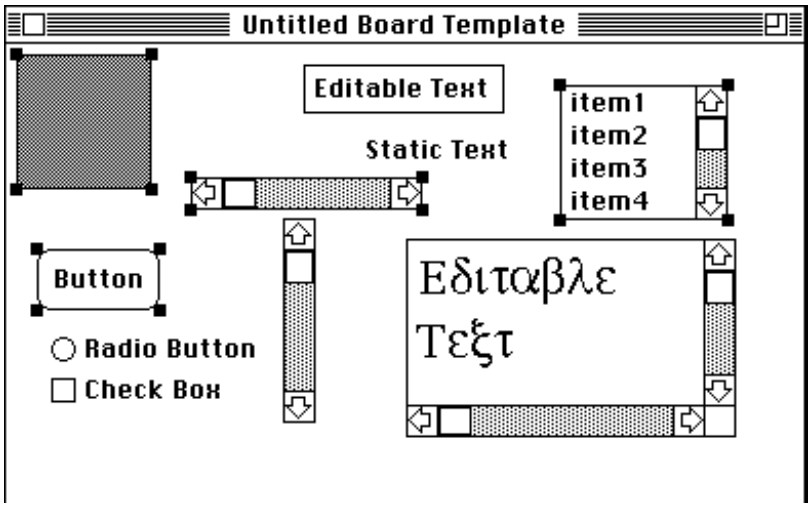

Figure 9.

L'application ScoreBoard

ScoreBoard est une application particulière réalisée à l'aide de *Board Editor* et destinée à la manipulation de structures musicales. Conçue de façon à rester le plus possible indépendante du formalisme ou du type de matériau expérimenté, *ScoreBoard* se présente sous la forme d'un Toplevel Lisp dans lequel des expressions symboliques peuvent être tapées et évaluées, et de boutons qui confèrent une interprétation aux dites expressions. *ScoreBoard* offre les caractéristiques suivantes:

- définition d'une structure musicale mono-voix arbitrairement complexe par la donnée de trois variables:
    - un patron métrique,
    - une liste de hauteurs ou d'agrégats de hauteurs,
    - un crible de durées;
- définition d'une structure polyphonique par la donnée des listes de ces valeurs;
- affichage des structures en notation musicale;
- navigation à l'intérieur de la partition. Insertion de nouvelles voix à la position courante. Effacement d'une voix;
- écoute des structures via un joueur Midi polyphonique ;
- fonctions de formatage élémentaires (mesures, systèmes) et d'impression.

La réalisation d'un protocole expérimental (un *générateur*) sous *ScoreBoard* se fait via l'écriture de fonctions Lisp quelconques dans leur forme, mais dont il est convenu qu'elles retournent un résultat interprétable selon les trois modes précités : patron métrique, liste de hauteurs ou crible de durées. *ScoreBoard* se charge alors automatiquement de toute la gestion périphérique (affichage en notation musicale, insertion dans la structure, génération du code Midi, association aux données rythmiques courantes, etc.) Un certain nombre de ces générateurs sont rassemblés dans une bibliothèque (*ScoreTools*), détaillée plus loin.

Le tableau de contrôle (figure 10) est composé de deux parties. La partie supérieure est un *Toplevel* Lisp. C'est ici que seront définies les structures musicales. La partie inférieure contient un ensemble de boutons, d'ascenceurs et d'indicateurs. Ici seront enregistrées et interprétées les structures définies dans le Toplevel.

Montrons en exemple le processus complet de création d'une voix musicale (figure 10 , figure 11). Tout d'abord, nous définissons un patron métrique par la sélection d'une expression symbolique dont l'évaluation retourne une liste de mesures, chacune sous la forme : (chiffrage divisions-du-temps), puis par une pression sur le bouton *metPat*.

Nous définissons ensuite un *crible d'impulsions*, c'est-à-dire une liste d'entiers croissants repérant des positions [32] dans le patron métrique considéré comme un flux ininterrompu d'impulsions. On procède ensuite comme précédemment, avec le bouton *pulse*.

La structure temporelle de notre voix musicale est maintenant complètement spécifiée. Nous définissons maintenant une *liste de hauteurs* qui vont être «accrochées» aux positions temporelles, grâce à une nouvelle expression symbolique et au bouton *pitch* [33] . Un agrégat est représenté par une liste de notes.

Le bouton *Compute Voice* calcule l'objet composite de hauteurs et de durées et l'affiche dans la fenêtre de visualisation (figure 11).

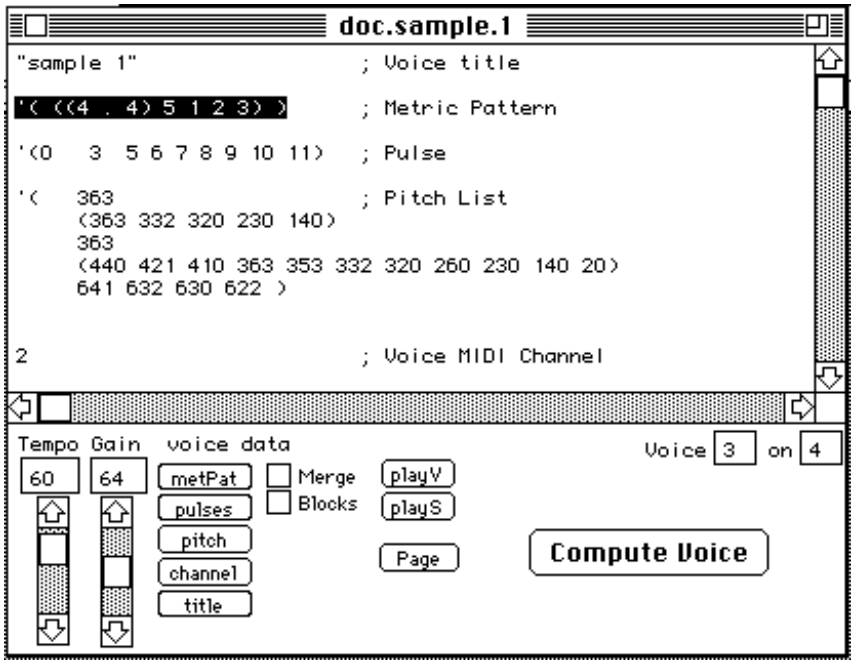

Figure 10.

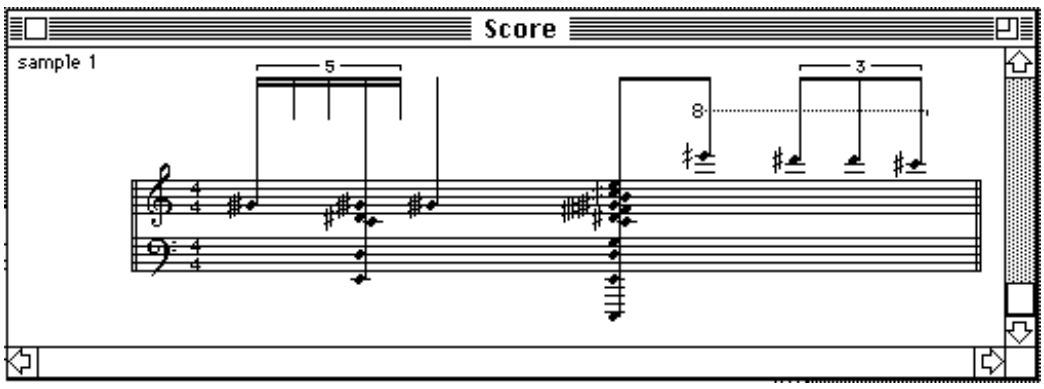

Figure 11.

ScoreTools. La librairie de générateurs

Un générateur est une fonction Lisp qui calcule une sous-structure musicale (liste de hauteurs, crible d'impulsion, patron métrique) et la ramène dans le format conventionnel décrit précédemment.

L'utilisation d'un générateur est simple : il suffit de taper un appel à la fonction concernée dans le Toplevel du tableau de contrôle, de sélectionner cet appel, puis d'appuyer sur le bouton adéquat.

Toute expression Lisp combinant des appels à des générateurs et construisant des listes avec un format interprétable comme liste de hauteurs, crible ou patron métrique peut être utilisée. Parmi les générateurs expérimentés, citons, dans le champ des hauteurs : spectres FM, spectres par description formantique de phonèmes, construction d'échelles avec LC [34] , fondamentales virtuelles, modes de distribution de hauteurs [35] , construction d'échelles par réseaux linéaires d'automates cellulaires, et, dans le champ des durées : construction de rythmes avec LC, construction de polyrythmies par dérivations de cribles [36] , etc.

Autres applications

D'autres applications ont été réalisées, dont l'utilisation conjointe avec ScoreBoard facilite la visualisation des données ou la génération de structures complexes. Citons par exemple *plotFM* et *plotVow* qui affichent des spectres FM ou vocaliques, *vEdit*, un éditeur formantique pour construire des spectres vocaliques (figure 12 , figure 13 , figure 14), *neuron*, un éditeur de réseau neuronal.

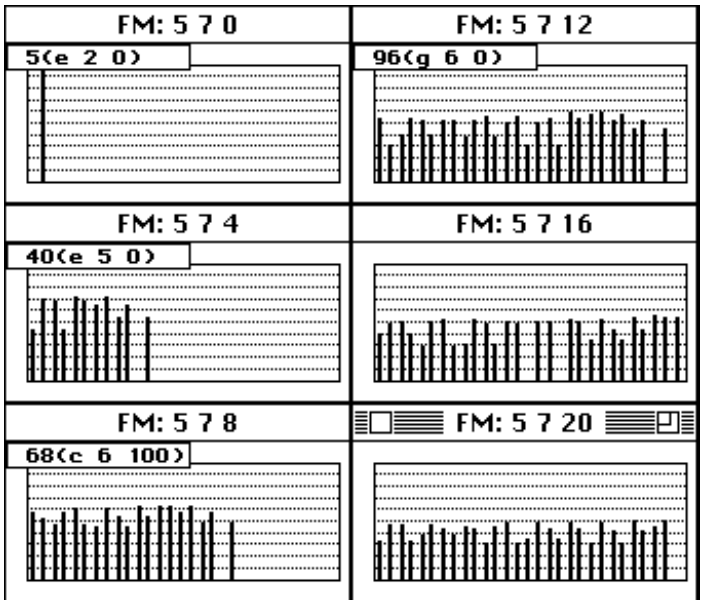

Figure 12.

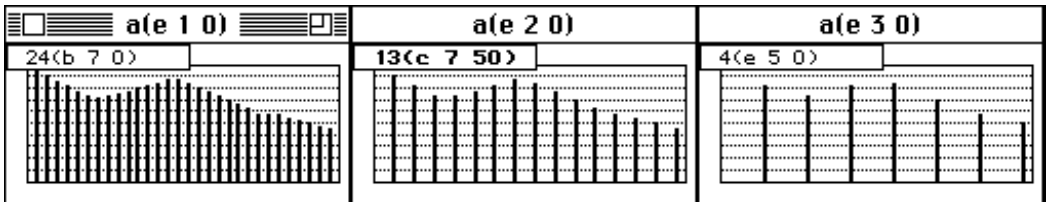

figure 13.

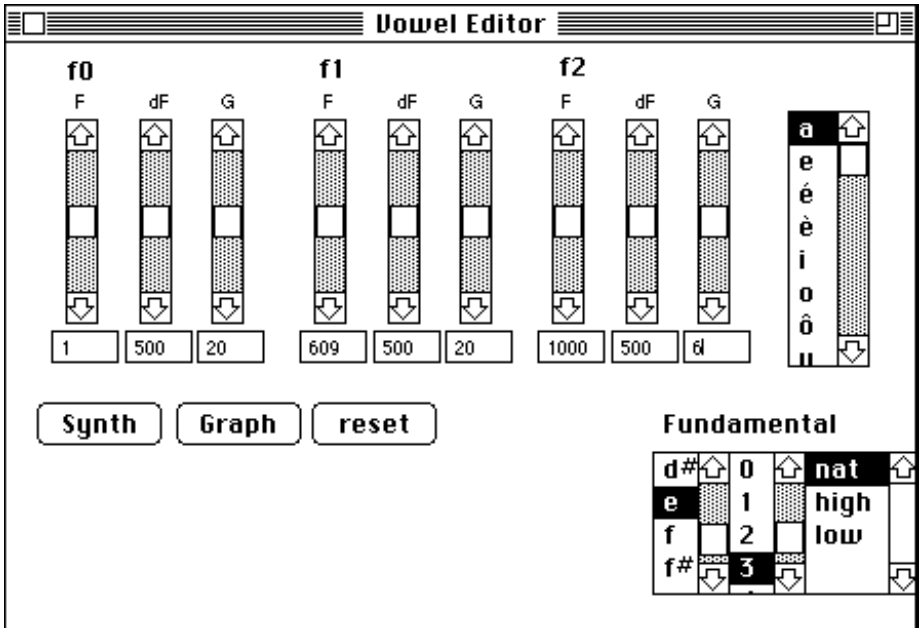

Figure 14.

Génération

L'application Génération décrite dans la section précédente appartient à une sous-classe de l'application ScoreBoard. Cela signifie que, par une simple instruction, nous créons, à partir d'un modèle existant, un nouveau modèle, plus spécifique et qui dispose de toutes les structures et de tous les opérateurs du précédent. Ainsi, dans Génération, on dispose des contrôles interactifs de ScoreBoard et il est possible, grâce au mécanisme de «sémantique libre» inhérent à Boards, de connecter en entrée et en sortie de cette application des calculs relevant de formalismes où d'environnements distincts. On pourrait par exemple calculer un ensemble de dérivations, puis projeter ce dernier sur une structure polyphonique calculée à partir de générateurs rythmique de la librairie ScoreTools. La connexion des formalismes a lieu précisément dans la *zone 1* (cf. figure 5) qui concentre l'héritage.

Références

H. Abelson, G. J. Sussman, *Structure and interpretation of Computer Programs.* MIT Press, Cambridge, 1985.

J.-M. Adrien, *Etude de structures complexes vibrantes. Application à la synthèse par modèles physiques.* Thèse de doctorat. Université Paris VI, 1989.

E. Amiot, G. Assayag, C. Malherbe, A. Riotte, *Duration structure generation and recognition in music writing.* Proceedings of the ICMC, La Haye, 1986.

G. Assayag, C. Malherbe, *Manipulation et représentation d'objets musicaux.* Communication aux Musical Editing & Printing Sessions, ICMC 94, Ircam, Paris, 1984.

G. Assayag, M. Castellengo, C. Malherbe, *Functional integration of complex instrumental sounds in music writing.* Proceedings of the ICMC, Vancouver, 1985.

Balaban et al, *Understanding Music with AI.* Ouvrage collectif, Balaban, Ebcioglu et Laske éditeurs, MIT Press, Cambridge MA, 1992.

A. Bonnet, C. Rueda, *Représentations symboliques de situations musicales.* Memo interne Ircam, 1992.

M. Chemillier, *Structure et méthode algébriques en informatique musicale.* Thèse de doctorat. LITP 90-4,

Université Paris VI, 1990.

M. Chemillier, *Langages musicaux et automates - la rationalité du langage sériel.* Actes du colloque Musique et assistance informatique, Marseille, 1990, pp. 211-227.

M. Chemillier, «Solfège, commutation partielle et automates de contrepoint»,*in Math.Inf.Sci.Hum.,* 110, 1990, pp. 5-25.

N. Chomsky, *Essais sur la forme et le sens.* Seuil, Paris, 1980.

J. Chowning, *«The synthesis of complex audio spectra by the mean of frequency modulation»,* in J.Aud.Eng.Soc. vol 21, nº.7, 1973.

J. Duthen, M. Stroppa, *Une représentation de structures temporelles par synchronisation de pivots.* Actes du colloque Musique et assistance informatique, Marseille, 1990, pp. 305-322.

S.-C. Kleene, *Mathematical Logic,* Wiley, New York, 1967.

M. Mesnage, A. Riotte, «Un modèle informatique d'une pièce de Stravinsky», *in Analyse Musicale,* 10. Paris, 1988, pp. 51-67.

M. Mesnage, A. Riotte, «Les variations pour piano opus 27: approche cellulaire barraquéenne et analyse assistée par ordinateur», *in Analyse Musicale,* 14. Paris, 1989, pp 51-66.

M. Mesnage, A. Riotte, «L'invention à deux voix nº.1, essai de modélisation informatique», *in Analyse Musicale,* 22. Paris, 1991, pp. 46-66.

B. Partee, «Montague Grammars and Transformational Grammars», *in Linguistic Inquiry* 6, 203-300, 1972.

P. van Hentenryck, *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

Y. Xenakis, *Musiques formelles.* Richard Masse, Paris, 1963.

NOTES

[1] Dans le cadre du département de la recherche musicale dirigée alors par Jean-Baptiste Barrière. Ce dernier a, par la suite, impulsé les principaux projets de CAO à l'Ircam.

[2] Plusieurs compositeurs, dont Marc-André Dalbavie et Kaija Saariaho, ont contribué par leur expertise à ce travail.

[3] Réalisé à l'Ircam par J. Duthen, M. Laurson et C. Rueda d'après un concept original de M. Laurson.

[4] Tristan Murail, qui fut un précurseur dans l'utilisation musicale de l'informatique personnelle, a fourni son expertise tout au long du travail de développement de PatchWork.

[5] Mikhaïl Malt a notamment exploré de manière systématique les conséquences de ce concept en analyse et en composition.

[6] Voir leur article dans ce numéro, pp. 43-60.

[7] Des modèles pour le contrepoint et la musique sérielle sont proposées dans Chemillier (1990a-c).

[8] Par exemple, le modèle de la première pièce pour quatuor à cordes de Stravinsky proposée par A. Riotte dès 1974 (Cf. Mesnage, 1988) ou celui des variations pour piano opus 27 de Webern par Riotte et

Mesnage (Cf. Mesnage, 1989).

[9] Cf. Chowning (1973) et Adrien (1989).

[10] Par exemple, les variantes à l'invention nº.1 à deux voix pour piano de Bach «composées» par A. Riotte à partir du modèle présenté dans Mesnage (1991).

[11] C'est le cas des itérateurs dotés d'une forte sensibilité aux conditions initiales. Voir l'exemple 2 à la fin de cet article.

[12] Cf. Chemillier (1990a).

[13] Il faut relativiser ce point de vue : en modélisation physique, on peut être amené à construire un modèle pour un objet unique ; par exemple, en acoustique des salles, lorsqu'on élabore un modèle informatisé d'un lieu particulier de diffusion existant ou projeté. Mais il ne faut pas confondre le modèle-théorie (ici, les lois mathématiques de diffusion, selon une option spéculaire ou diffuse) et le modèle-maquette, objet réduit reproduisant fidèlement le comportement de l'original. Le modèle informatique d'une oeuvre en devenir serait une sorte de version progressive de cette dualité, opérant un va-et-vient constant entre les deux termes.

[14] Sur la formalisation des polyrythmies, voir Amiot (1986). M. Balaban (1989, pp 110-138) propose une formalisation permettant de rendre compte simultanément des aspects temporels et hiérarchiques des structures musicales. B. Bel (Balaban, 1989, pp 64-109) propose une représentation des structures rythmiques dans le cadre des grammaires formelles, ainsi qu'une méthode pour synchroniser des objets sonores dont les relations temporelles sont spécifiées de manière incomplète. Un modèle de description des structures temporelles posant le problème de la synchronisation est décrit dans Duthen (1990).

[15] Concept dû à Xenakis. Cf. Xenakis (1963).

[16] Voir, par exemple, dans Abelson (1985) les différents modèles d'évaluation du langage Lisp ; les modèles par substitution ou métacirculaires décrivant des invariants, le modèle par machine à registre décrivant le comportement dans le temps. L'exploit est ici que tous ces modèles sont écrits dans le langage même qu'ils modélisent.

[17] Cf. Kleene (1967).

[18] Notamment dans les grammaires de Montague, décrites dans Partee (1972) et discutées dans Chomsky (1980).

[19] En informatique, un type définit une classe de données simples (par exemple les nombres entiers ou les symboles) ou composées (obtenue par combinaison de types primitifs, par exemple le type note pourrait être défini par la combinaison des types hauteur, durée, intensité) ; un type nous informe notamment sur la structure des objets qui lui appartiennent et sur les opérateurs qui peuvent le manipuler. Pour plus de détails, voir Abelson (1985).

[20] Un langage de programmation orienté objet permet la création d'entités dynamiques (les objets) à partir de prototypes conceptuels (les classes, organisées en hiérarchies et dont les propriétés se propagent par héritage). Les objets communiquent par des messages dont l'interprétation est associée aux classes par des procédures appelées méthodes.

[21] Cf. Chemillier (1990a).

[22] Pour une synthèse des questions, très étudiées aujourd'hui, de CSP (Constraint Satisfaction Problems), voir Van Hentenrink (1989). Pour une discussion de leur application musicale, voir Bonnet (1992)

[23] Une technique qui consiste à remplacer la programmation textuelle traditionnelle par des assemblages

schématiques (les patches) manipulés grâce à une interface graphique interactive. Voir Glinert (1990).

[24] Pour reprendre la jolie expression de Tristan Murail. Plus techniquement, ce problème de tolérance des opérateurs aux variations du type des objets qu'ils traitent est connu en informatique sous le nom de généricité ou polymorphisme. Cf. Abelson (1980).

[25] Par exemple a des pôles harmoniques : cf. Assayag (1985).

[26] Cette logique de processus généralisée à des états complexes a été expérimentée dans le champ instrumental par des compositeurs comme K. Saariaho, T. Murail ou M.-A. Dalbavie.

[27] Une discussion complète du formalisme ébauché ici, contenant notamment une justification topologique de la notion de région, a été rédigée par A. Champert et P. Haïm.

[28] Le_Lisp 15.21 INRIA/Act.

[29] Voir définition plus haut.

[30] Un Toplevel (ou écouteur) Lisp est une fenêtre dans laquelle l'utilisateur peut interactivement taper et évaluer des expressions symboliques.

[31] L'interface du logiciel Boards même a été réalisée grâce à Board Editor par un mécanisme de «boot-strap», bien connu dans la compilation des langages informatiques.

[32] Les cribles on été introduits par Xenakis, qui les a principalement appliqués au champ des hauteurs. Cf. Xenakis (1963).

[33] Les hauteurs sont codées sur un nombre dont les chiffres successifs représentent un numéro d'octave, un degré diatonique et une altération.

[34] Langage de construction de cribles par expressions algébriques réalisé par l'auteur à l'Ircam, avec C. Malherbe. Cf. Amiot (1986).

[35] Mécanisme mis au point par C. Malherbe pour régler le choix et l'ordonnancement temporel d'éléments pris dans une échelle. Cf. Assayag (1984).

[36] Mécanisme proposé par C. Malherbe. Cf. Amiot (1986).