13

Objets et musique

CE CHAPITRE DÉCRIT QUELQUES APPLICATIONS typiques de la programmation par objets à la réalisation de systèmes mettant en œuvre des connaissances musicales. On montre comment les mécanismes de base de la programmation par objets permettent de représenter tout un ensemble de concepts de base de la musique tonale. Des mécanismes spécifiques sont ensuite utilisés pour résoudre divers problèmes : règles pour l'analyse, satisfaction de contraintes pour l'harmonisation, propagation de contraintes pour l'édition de partitions. Ces applications musicales posent des problèmes de représentation de connaissances difficiles dans un cadre concret, mais dont la portée dépasse largement le cadre de la musique.

13.1 Introduction

Il existe une profonde analogie entre musique et calcul (séquence, répétition, branchement conditionnel). Cette analogie explique sans doute pourquoi l'ordinateur a été utilisé pour « faire de la musique » sous toutes ses formes, que ce soit pour la produire, l'analyser ou la simuler. Cette situation fournit de fait un cadre expérimental dans lequel les représentations de connaissances musicales peuvent être directement opératoires, et donc, dans un certain sens, validées, au contraire d'autres champs artistiques.

Les connaissances mises en œuvre en musique sont complexes : à la fois abstraites (intervalles et accords), définies de manière informelle (par exemple les contours flous des *motifs*) et faisant l'objet de typologies sophistiquées et incomplètes (par exemple celle des formes musicales : du rondo à la sonate). La musique offre donc un terrain d'expérimentation privilégié pour l'Intelligence Artificielle, comme l'ont déjà montré les travaux de Hofstadter sur la réflexivité et la musique [Hofstadter, 1985], de Minsky sur le sens commun musical [Minsky et Laske, 1992], ou de Greussay sur l'analyse [Greussay, 1985] (voir [AFIA, 1995] pour un panorama français sur cette question). Ce lien entre musique et intelligence artificielle est particulièrement présent dans le domaine des objets. C'est sur cet aspect que nous nous concentrons dans ce chapitre.

13.1.1 Musique et objets

La musique a été pensée naturellement en termes d'objets depuis ses premières formalisations, ceux de Pierre Schaeffer [Schaeffer, 1966] en étant une incarnation particulièrement radicale et concrète. Il n'est donc pas étonnant que les techniques de la programmation par objets aient été appliquées à de nombreux problèmes musicaux. Bien sûr, il n'y a pas toujours concordance entre les objets pensés par les musiciens et les objets de nos langages, mais ce décalage est moteur et donne lieu bien souvent à des modélisations qui s'avèrent être des sources d'inspiration fructueuses pour d'autres domaines.

En France, le système FORMES [Rodet et Cointe, 1991], un précurseur des systèmes d'aide à la composition par les techniques à objets, était basé sur la notion de *processus*, regroupant un objet au sens classique du terme, et un moniteur pour son ordonnancement dans le temps et la synchronisation de ses sous-objets. Ce système fut utilisé principalement pour des compositions musicales, mais il est général et applicable à d'autres contextes comme l'animation ou la synthèse de la parole. Ralph Johnson, un des auteurs des travaux sur les *frameworks* et les *design patterns* [Gamma *et al.*, 1994] (voir chapitre 4), a travaillé dans le groupe de recherche de Carla Scaletti sur le système à objets *Kyma* [Scaletti, 1987], un des premiers environnements interactifs pour la composition et la synthèse sonore, aujourd'hui commercialisé avec une carte d'acquisition spécialisée pour le traitement du signal temps réel. Les travaux de Steven Pope abordent depuis une dizaine d'années divers problèmes musicaux avec les techniques à objets : éditeurs de partitions, environnements de composition algorithmique en temps réel et outils d'assistance à l'exécution musicale. Ces travaux sont incarnés dans le système MODE [Pope, 1991a]. Plus récemment, Bill Walker a développé un système d'improvisation musicale par objets fondé sur l'idée que l'improvisation est une forme de conversation : le système *ImprovisationBuilder* [Walker *et al.*, 1992]. De manière générale, le lecteur pourra consulter [Pope, 1991b] qui décrit quelques applications importantes de la programmation par objets à la musique, ainsi que le numéro spécial du *Computer Music Journal* sur les objets [Pope, 1989].

13.1.2 La musique comme prétexte : le système MUSES

Nous décrivons ici quelques problèmes musicaux typiques et des solutions utilisant les techniques à objets : la musique est dans ce cadre considérée comme prétexte pour poser des problèmes de représentation fondamentaux qui dépassent le cadre de la musique, comme des problèmes de représentation du temps, d'ontologies réutilisables ou d'expression de connaissances. Plus précisément, nous prendrons comme exemples un certain nombre de travaux effectués au LIP6 (Laboratoire d'Informatique de Paris 6), autour du système MUSES, qui sert de base à toute une série d'expérimentations en représentation de connaissances musicales. MUSES se présente sous la forme d'une bibliothèque de classes représentant les concepts fondamentaux de la musique tonale [1] : *pitch-classes*, notes, accords, gammes, mélodies.

1. C'est-à-dire la musique basée sur la tonalité comme la musique classique, le jazz, le rock, etc. par opposition à la musique post-tonale comme le dodécaphonisme ou la musique électro-acoustique.

La taille de cette bibliothèque est d'environ 100 classes et 1500 méthodes [Pachet, 1994a]. Plusieurs applications abordant des problèmes musicaux concrets ont été construites autour de MUSES (voir figure 13.1) : un système d'analyse harmonique automatique [Mouton et Pachet, 1995] [Pachet, 1994c], un système d'harmonisation de chorals à quatre voix [Pachet et Roy, 1995a] [Pachet et Roy, 1995b], un système de simulation d'improvisations [Ramalho et Ganascia, 1994] [Ramalho et Pachet, 1994] et un système d'induction automatique de motifs mélodiques [Rolland et Ganascia, 1996].

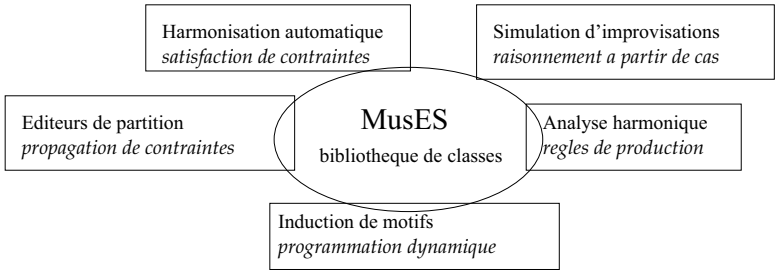

FIG. 13.1: *Le système* MUSES *et ses principales applications.*

Dans ce chapitre, nous décrirons quelques uns des problèmes soulevés par ces systèmes écrits en MUSES : représentation des altérations (dièses et bémols) en utilisant le polymorphisme, représentation de structures temporelles, représentation de règles d'analyse et de contraintes pour l'harmonisation et problèmes d'éditions de partitions.

13.2 MUSES : concepts de base et programmation par objets

MUSES est une bibliothèque de classes représentant des connaissances consensuelles sur la musique tonale. Il est bien entendu qu'il n'existe probablement pas de représentation universelle des notions de note, accord, gamme, arpège, etc. Mais il est possible de trouver des représentations raisonnablement réutilisables pour ces concepts de base dans un cadre restreint d'applications musicales. Nous nous sommes essentiellement concentrés sur des applications résolvant des problèmes à caractère analytique, et portant sur des corpus de musique occidentale tonale, à tempérament égal (c'est-à-dire reposant sur le clavier bien tempéré). Dans ce contexte, les premiers concepts à représenter sont ceux de note (indépendante de l'octave ou *pitch-class*) et d'altération (les dièses, bémols et bécarres, signes utilisés pour modifier la hauteur d'une note). Le problème est de représenter non pas tant le signe lui-même, mais bien son intension harmonique. Ainsi, un Ré dièse sonnera-t-il exactement comme un Mi bémol en musique tempérée : c'est la même touche sur le piano. Mais les deux notes portent des intensions musicales bien différentes,

`NoteNaturelle<diese`
`"rend la variable d'instance diese"`
`^diese`
	`NoteBemol<diese`
	`"rend la naturelle correspondante"`
	`^naturelle`

`NoteDiese<diese`
`"rend la variable d'instance diese"`
`^diese`
	`NoteDoubleBemol<diese`
	`"rend la naturelle bemolisee"`
	`^naturelle bemol`

FIG. 13.2: *Les quatre implémentations en* SMALLTALK *de la méthode* `diese`*, représentant une partie de l'algèbre des altérations.*

aussi différentes que le sens des mots « sang », « sans », « s'en », ou « cent »[2]. Ces altérations forment alors une sorte d'algèbre : les dièses et les bémols s'annulent réciproquement. Mais cette algèbre n'est pas simple. Ainsi, une note peut être diésée, voire double-diésée, mais pas plus. Par ailleurs, les notes entretiennent une relation d'équivalence, celle des hauteurs : ré dièse et mi bémol sont des notes distinctes dans la pensée, mais elles *sonnent* pareil, elles représentent la même hauteur.

Une manière particulièrement satisfaisante de représenter ces altérations est d'exploiter le mécanisme de polymorphisme, qui est à la base des langages de programmation par objets. Ceci se fait en considérant une altération comme une méthode et les notes comme des instances de classes différentes, suivant leur altération [Pachet, 1994b]. On définit ainsi six classes de notes : `NoteNaturelle`, ayant 7 instances représentant les notes non altérées, `NoteAlteree`, classe abstraite ayant quatre sous-classes: `NoteDiese`, `NoteBemol`, `NoteDoubleDiese` et `NoteDoubleBemol`, représentant les différents types de notes altérées. L'altération dièse est représentée comme une méthode polymorphe, ayant 4 implémentations différentes (voir figure 13.2). Le type du résultat de cette méthode dépend de la classe dans laquelle elle est implémentée. Ainsi, la méthode `diese` de la classe `NoteNaturelle` rend une instance de `NoteDiese`, calculée statiquement et conservée dans une variable d'instance de même nom ; celle de la classe `NoteDiese` une instance de `NoteDoubleDiese`, aussi conservée dans une variable d'instance. Pour la classe `NoteBemol`, la méthode renvoie la note naturelle correspondante (bémol et dièse s'annulent). La classe `NoteDoubleBemol` renvoie la note naturelle « bémolisée » (double bémol + dièse = bémol). Il faut noter que la méthode `diese` n'est pas implémentée dans la classe `NoteDoubleDiese`. Cette absence provoquera donc une erreur si l'on tente de diéser une note double dièse, ce qui est conforme à notre cahier des charges : les triples dièses n'existent pas ! Le même mécanisme s'applique bien sûr aux bémols.

Enfin, les notes altérées héritent toutes d'une classe abstraite `NoteAlteree`, qui implémente la notion de bécarre (l'anti-altération) par un pointeur vers la note naturelle correspondante (voir figure 13.3).

Une fois la théorie des *pitch-class* et des altérations correctement représentée, l'échafaudage peut être solidement construit. La première notion ajoutée est celle de note effective, dépendante de l'octave (`OctaveDependentNote`: Do6, Do5, etc.). Une solution naturelle serait de considérer les *pitch-class* comme des classes — au sens des langages de classes — et les `OctaveDependentNote` comme des ins-

2. La notion d'altération pose uniquement problème en musique tonale ; la musique post-tonale reléguant l'altération à un signe purement local, dénué de toute intension harmonique.

tances de ces classes. Les différentes classes de *pitch-class* deviendraient alors des métaclasses. Mais ceci nécessite un langage permettant de définir ses propres métaclasses, comme CLOS [Kiczales *et al.*, 1991], ou CLASSTALK [Briot et Cointe, 1989 ; Rivard, 1997]. La solution adoptée en MUSES est l'agrégation : `OctaveDependentNote` est une classe qui agrège une *pitch-class* et un numéro d'octave.

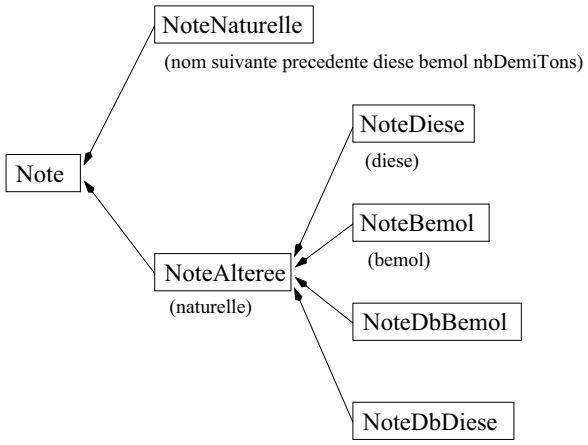

FIG. 13.3: *La hiérarchie des classes de notes et la représentation des altérations en* MUSES. *Les variables d'instance sont entre parenthèses.*

Des notions plus complexes sont ensuite ajoutées, comme les intervalles et leur typologie, qui doivent aussi respecter l'enharmonie (le rapport entre quartes augmentées et quintes diminuées par exemple), les gammes, les accords, etc. Des notions plus abstraites comme celle d'*Analyse* sont aussi représentées. Une fois ces concepts de base correctement définis, des méthodes permettent par exemple de calculer la liste des tonalités possibles pour chaque accord, fournissant ainsi une base solide pour construire des systèmes d'analyse. Ainsi du calcul des tonalités possibles de Do majeur :

```
(Chord fromString: 'C maj') possibleTonalities  -->
([V of F HungarianMinor] [VI of E HungarianMinor]
 [IV of G MelodicMinor] [V of F MelodicMinor] [I of C Major]
 [IV of G Major] [V of F Major] [V of F HarmonicMinor]
 [VI of E HarmonicMinor])
```

13.3 Les objets temporels de MUSES

Une fois le cas des objets statiques traité, se pose le problème de la représentation des objets temporels. De nombreux travaux en intelligence artificielle sont consacrés à la représentation du temps. Ils peuvent être regroupés en fonction du type de primitive temporelle sur laquelle ils reposent : intervalle [Allen, 1984], points [McDermott, 1982] ou événements [Kowalski et Sergot, 1986]. Formellement, les trois

approches peuvent être reliées les unes aux autres et permettent de représenter les mêmes connaissances [Tsang, 1987]. Cependant, le choix effectué reflète la dimension du temps que l'on considère comme fondamentale : la durée (intervalle) ou l'instant (point, événement). La représentation par intervalle est particulièrement adaptée à l'idée même de note musicale. On peut alors distinguer deux manières principales de représenter les objets temporels, suivant le choix adopté pour représenter ces intervalles : 1) le temps est représenté *en dehors des objets*, dans des collections temporelles, c'est le choix du système MODE, et 2) le temps est *dans les objets* eux-mêmes, c'est le choix adopté pour le système MUSES.

13.3.1 Le temps en dehors des objets

Le système MODE [Pope, 1991a] propose une représentation du temps basée sur la notion d'événement (`Event`), classe abstraite représentant des processus temporels. Cette classe possède une variable d'instance `duration`, représentant la durée du processus, dans une unité arbitraire, elle-même réifiée. Par ailleurs, les événements peuvent avoir un certain nombre de propriétés, accessibles par un dictionnaire. Le point important — et astucieux — est que ces événements temporels ne connaissent pas leur temps initial. La justification conceptuelle est qu'un objet temporel n'a de sens qu'au sein d'une collection temporelle déterminée. La durée est ainsi un attribut intrinsèque, alors que le temps de début est vu comme extrinsèque. Ceci permet en pratique de distribuer ces objets dans plusieurs collections temporelles et de faciliter un certain nombre de tâches d'édition (copier/coller). `Event` étant une classe abstraite, une hiérarchie de classes concrètes est construite pour représenter les événements courants comme les notes de musique.

La deuxième notion de base est celle d'`EventList`, qui représente une collection d'événements. Cette collection est constituée de paires (`durée`, `Event`), ou `durée` représente le temps initial de l'événement en question, exprimé comme une durée par rapport au début de la collection temporelle. Cette représentation permet de dissocier le temps initial de l'objet temporel, favorisant ainsi la modularité. En outre, elle permet de représenter, récursivement, les listes temporelles comme des événements, à moindre frais : `EventList` est tout simplement une sous-classe de `Event` ! Ainsi, une `EventList` peut elle-même être considérée comme un objet temporel, à l'intérieur d'une autre collection temporelle, ce qui permet de créer des structures temporelles arborescentes (voir figure 13.4). Noter que ce schéma est une application directe du *design pattern Composite* décrit dans [Gamma *et al.*, 1994].

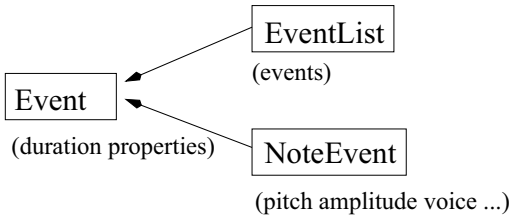

FIG. 13.4: *La hiérarchie des classes d'objets temporels en MODE.*

13.3.2 Le temps dans les objets

La deuxième approche, celle suivie dans le système MUSES [Pachet *et al.*, 1996], consiste à créer des structures d'objets temporels explicites. Cette approche est justifiée par le besoin pour certaines applications comme les systèmes d'analyse, de connaître pour chaque note sa position dans une mélodie. Pour ces applications, la représentation précédente est alors inadaptée. Dans cette seconde approche, les objets temporels sont représentés à l'aide de trois notions de base :

- `Lapse`, représentant un intervalle de temps et déterminé par deux variables d'instance ici réconciliées (temps initial, durée),
- `TemporalObject`, représentant un objet temporel, avec une variable d'instance (`lapse`) pointant vers une instance de `Lapse`,
- `TemporalCollection`, représentant une collection d'objets temporels, triés par temps initial croissant.

Comme nous l'avons vu plus haut, MUSES contient des représentations d'un certain nombre de concepts atemporels (pitch-classes, intervalles, accords, etc.). Ces objets pour la plupart ont des correspondants temporels. Une solution pour les représenter serait de définir ces objets temporels en les faisant hériter à la fois de la classe abstraite `TemporalObject` et des classes d'objets intemporels. La solution adoptée en MUSES consiste à utiliser le mécanisme bien connu dans les langages à objets de *délégation* [Lieberman, 1986], ce qui permet, entre autres, d'éviter l'emploi de l'héritage multiple (voir *chapitre 8*). Nous introduisons de plus une notion supplémentaire, permettant de mettre en œuvre cette délégation de manière générique : celle d'« enveloppe temporelle » (`TemporalObjectWrapper`). Cet objet possède une variable d'instance pointant vers un objet non temporel, et lui délègue toutes les opérations non temporelles. Nous faisons la même chose pour les classes de collections (`TemporalCollectionWrapper`). Nous obtenons donc *in fine* un schéma à cinq classes (cf. figure 13.5).

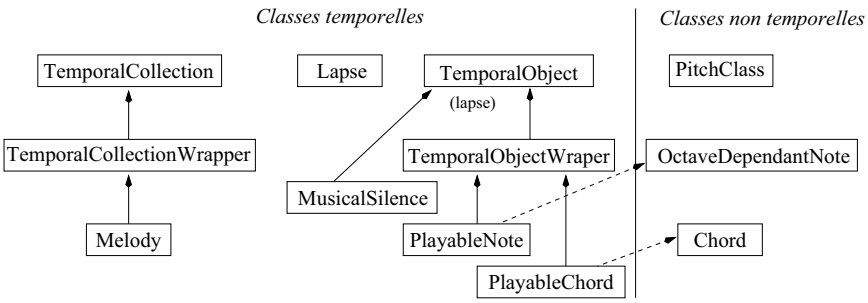

FIG. 13.5: *La hiérarchie des classes temporelles en* MUSES. *Les flèches pleines représentent l'héritage de classe, les traits pointillés la délégation.*

13.4 Analyse harmonique de grilles de jazz

Une des activités principales du musicologue est l'*analyse* de pièces musicales, dans le but de mieux comprendre un style, de caractériser un compositeur, ou d'expliciter des structures musicales cachées. Plusieurs théories pour l'analyse ont été développées : stylistique, harmonique, tonale, formelle, Schenkerienne, etc. La plupart de ces théories ont la caractéristique d'être particulièrement bien formalisées, et de nombreuses tentatives d'informatisation de ces techniques ont été proposées [Pachet, 1998]. L'analyse harmonique est un problème particulièrement intéressant en représentation de connaissances car les règles sous-jacentes à cette analyse font l'objet d'un consensus. Nous l'illustrons ici sur l'exemple de l'analyse harmonique de séquences d'accords de jazz.

13.4.1 Deux *patterns* de règles pour l'analyse

Étant donné une séquence d'accords, le problème consiste à déterminer la tonalité de chaque accord dans la séquence. Cette tonalité dépend à la fois de la structure de l'accord lui-même (accord majeur, mineur, de septième, etc.), de sa position dans la séquence (temps fort, temps faible) et des accords voisins. Le but de l'analyse est de trouver des tonalités qui soient plausibles musicalement. Cette plausibilité musicale fait appel à des règles harmoniques précises et bien connues qui permettent de décrire des formes musicales de différents niveaux. Par exemple, on peut décrire avec précision une *résolution*, comme un accord de septième suivi d'un accord parfait (du moins dans sa version simple). De même, une structure dite *AABA* peut se décrire par une répétition de quatre formes vérifiant des contraintes d'identité simples. Enfin, une bonne analyse tente de trouver des tonalités qui soient communes au plus grand nombre d'accords possibles (principe de minimisation).

Plusieurs approches ont été explorées pour l'analyse harmonique par ordinateur : procédurales [Ulrich, 1977], à base de grammaires génératives [Steedman, 1984], de grammaires systémiques [Winograd, 1968], de systèmes experts [Maxwell, 1992]. Leur principal inconvénient est de représenter toutes les connaissances mises en jeu à l'aide d'un seul paradigme. Comme toujours, ce paradigme, quel qu'il soit, est souvent bien adapté à un type de connaissances, mais pas à d'autres. On aboutit ainsi à des systèmes complexes, difficiles à comprendre, et limités dans leurs performances. L'approche que nous avons suivie consiste à trouver un modèle de l'analyse qui exploite les structures objets existantes et qui soit généralisable à d'autres problèmes d'analyse, pas forcément musicaux. Cette approche est fondée sur un modèle du processus d'analyse qui relève de la psychologie de la forme (*gestalt*) : on identifie dans un premier temps des formes musicales classiques, comme les II-V, les anatoles, cadences, résolutions ou autres marches harmoniques. Par exemple, un II-V sera reconnu lorsqu'on trouve deux accords voisins tels que la tonique du second soit la quinte de la tonique du premier, que le premier est mineur, le deuxième de septième, etc. (par exemple, la séquence D min 7 / G 7). Dans un deuxième temps, on tente de regrouper les accords isolés à l'aide de règles de regroupement. Par exemple, une règle dit que si un accord isolé est voisin d'une forme analysée et que la tonalité de cette forme analysée fait partie des tonalités possibles de cet

accord, alors on peut analyser l'accord dans cette tonalité, et donc grouper l'accord et la forme pour créer une forme englobant les deux (par exemple, un accord isolé de C majeur, suivi d'une longue séquence d'accords analysée en C majeur).

Dans les deux cas, les règles spécifient des manières d'agréger des formes d'ordre n pour créer des formes d'ordre $n + 1$. L'analyse est terminée lorsqu'une forme couvrant l'ensemble de la séquence a été trouvée [Mouton et Pachet, 1995][Pachet, 1994c]. Pour implémenter ces règles, le formalisme des règles de production est particulièrement bien adapté. Nous avons montré [Pachet, 1998] que les règles d'agrégation, que ce soit pour les formes simples (exemple de la règle de II-V) ou pour les règles de regroupement générales (accord isolé suivi de forme analysée), suivent toutes le même schéma :

```
 Soient a1 a2 des instances (d'une sous-classe) de FormeMusicale
SI certaines propriétés sur l'agencement de a1 et a2 sont vraies,
ET certaines propriétés sur leurs caractéristiques locales sont vraies
ALORS créer une instance d'une sous-classe de FormeMusicale
                         agrégeant les deux formes
```

En outre, un des points importants de cette approche, notamment par rapport aux modèles à base de grammaire, est de permettre l'écriture de règles de destruction de formes, appelées *règles d'oubli*, qui permettent d'assurer une convergence rapide du raisonnement, en limitant dynamiquement le nombre de formes créées. Les problémes de convergence (risque de recréer des formes détruites, et donc de boucler) sont résolus en adoptant des stratégies particulières de contrôle, à base d'agendas [Dojat et Sayettat, 1994]. Ces règles suivent, elles, le schéma suivant :

```
 Soient a1 a2 des instances de (une sous-classe de) FormeMusicale
SI certaines propriétés sur l'agencement de a1 et a2 sont vraies,
ET certaines propriétés sur leurs caractéristiques locales sont vraies
ALORS supprimer a2
```

13.4.2 Discussion : Patterns et Frameworks

Le modèle élaboré pour rendre compte du processus d'analyse, outre ses intérêts purement musicologiques, présente plusieurs intérêts pour la programmation par objets, et particulièrement pour l'étude des *patterns* et des *frameworks*. D'une part nous avons mis en évidence deux *patterns* de règles, au sens de [Gamma *et al.*, 1994]. Ces deux *patterns* permettent à eux seuls de représenter la totalité des bases de règles nécessaires au raisonnement. Par ailleurs, une comparaison de ce système avec un système développé dans le domaine médical — le système NéoGanesh [Dojat et Sayettat, 1994] — a permis de mettre en évidence une forte analogie entre les deux types de raisonnements (médical et musical), conduisant ainsi à un *framework* permettant la représentation de raisonnements temporels hiérarchiques [Pachet et Dojat, 1995]. Enfin, la mise en évidence d'un problème de circularité dans les grilles de jazz (la fin de la grille précède en fait le début), a été le point de départ d'une recherche plus formelle sur une extension des intervalles de Allen aux intervalles circulaires [Pachet et Carrive, 1996].

13.5 Harmonisation automatique

Le problème de l'harmonisation automatique (le plus souvent de chorals à quatre voix comme en a composé Bach), est d'une certaine manière le problème dual de celui de l'analyse. Il consiste, pour une mélodie donnée, à écrire trois autres voix, de manière à respecter les règles de l'harmonie et du contrepoint. Les règles sont du style : « l'intervalle de quarte augmentée est interdit entre deux notes consécutives », ou « la sensible doit monter à la tonique », ou encore « les intervalles de quintes entre deux voix ne doivent pas être parallèles ».

Ce problème a déjà été abordé par diverses techniques, dont celles de la satisfaction de contraintes (CSP, voir *chapitre 9*) : [Ovans et Davison, 1992] par la CSP pure, [Ebcioglu, 1992] utilisant les algorithmes de retour-arrière intelligent, [Tsang et Aitken, 1991] par la programmation logique avec contraintes, [Ballesta, 1994] avec une combinaison de contraintes et d'objets. Aucune de ces approches n'a permis de construire de systèmes aux performances réalistes (une minute environ pour le plus rapide d'entre eux, sur des mélodies d'une dizaine de notes, alors qu'un bon musicien résout ces problèmes en un temps quasiment instantané !).

Dans notre contexte, ce travail constitue un remarquable terrain d'investigation pour les systèmes intégrant satisfaction de contraintes et objets (voir à ce sujet *chapitre 9*). L'exploitation des structures objets complexes comme celles de MUSES permet en effet de poser le problème de manière plus claire et de le résoudre de manière plus satisfaisante. Les connaissances liées aux objets de base (notes, accords, intervalles) sont représentées de manière procédurale avec la bibliothèque MUSES. Les lois des traités d'harmonie et de contrepoint sont exprimées comme des contraintes portant sur ces objets, et sont gérées par un résolveur de contraintes général. On peut montrer alors que l'exploitation des structures objets dans la résolution du problème permet non seulement d'augmenter la lisibilité des contraintes, mais aussi de réduire considérablement leur complexité et donc le temps d'exécution [Pachet et Roy, 1995a] [Pachet et Roy, 1995b]. Un des points-clé de cette approche est de remplacer des contraintes d'arité élevée portant sur des objets atomiques (des valeurs), par des contraintes d'arité faible, portant sur des objets complexes. Typiquement, la fameuse règle interdisant les « quintes parallèles » entre accords consécutifs s'exprime dans l'approche standard par une contrainte (complexe) d'arité huit (quatre notes par accord, chaque note étant considérée ici comme un objet atomique, par exemple une hauteur), et dans la nôtre par une contrainte d'arité deux (deux objets accords). Le système ainsi obtenu est non seulement plus lisible et modifiable, mais aussi environ 10 fois plus rapide que tous les autres.

13.6 Éditeurs graphiques

La conception d'éditeurs graphiques constitue un terrain privilégié de la programmation par objets. Le problème de l'édition de partitions est un cas particulièrement difficile. Plusieurs types de logiciels sont proposés sur le marché, répondant à divers besoins, souvent inconciliables. D'une part des logiciels permettent de faire de l'édition de partitions à proprement parler. Ceux-ci offrent à l'utilisateur

toute la palette des signes graphiques musicaux, avec laquelle celui-ci peut composer sa partition comme un dessin. D'autres logiciels permettent une édition graphique plus limitée mais interactive, obtenue à partir d'une représentation sémantique riche des notes, comme les logiciels de *sequencers* par exemple. On retrouve alors le même dilemme que pour les traitements de texte : interactivité et automatismes, contre *batch* et flexibilité (par exemple Word contre LaTeX).

La complexité des objets mis en jeu dans ces éditeurs peut être illustrée par un exemple typique : l'affichage des notes et la gestion des syncopes. La représentation graphique d'une note dépend à la fois de sa durée (il y a des blanches, des noires, des croches, etc.) mais aussi de son emplacement, de sa position dans la mesure. Ainsi, une note durant 4 temps, dans une mesure à 4 temps et commençant sur le premier temps doit s'afficher par une *ronde*, graphiquement représentée par un rond sans queue, illustrée figure 13.6 (à gauche)[3]. Lorsque cette même note commence sur le deuxième temps, on ne peut plus l'afficher comme telle, pour deux raisons : d'abord la mesure ne peut contenir que 4 temps, et donc la note « dépasse », et ensuite la règle de syncope stipule qu'une note ne peut franchir un temps plus fort que son temps de départ. Ainsi, la notation correcte est celle d'une noire liée à une blanche (syncope au sein de la mesure), liée encore à une noire à la mesure suivante (cf. figure 13.6, milieu). Si cette même note commence sur le deuxième quart du premier temps, les mêmes contraintes conduisent à la représentation encore plus complexe de la figure 13.6 (à droite). Ces trois représentations doivent être recalculées rapidement, car les notes peuvent être déplacées à la souris. Le problème est encore plus complexe quand les mesures sont de taille variable.

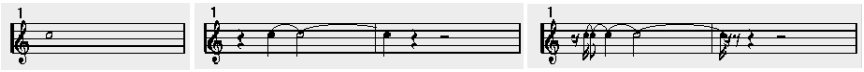

FIG. 13.6: *Trois représentations graphiques pour la même note dans l'éditeur* MUSES.

Trouver une conception par objets d'un tel éditeur qui permette d'assurer un bon compromis entre interactivité et calculs automatiques est un problème ouvert. De nombreuses recherches exploitent les formalismes de *propagation de contraintes* pour proposer des *frameworks* dans lesquels de tels éditeurs sont, en théorie, plus faciles à construire [Vlissides et Linton, 1990] [Brant, 1995] (voir aussi *chapitre 9*). L'application de ces techniques à des éditeurs de partitions n'est pourtant pas facile. Comme nous l'avons vu, la représentation graphique d'une même note peut donner lieu à la création d'un ou de plusieurs objets graphiques, et ce, dynamiquement. Or les formalismes de contraintes ne s'accommodent pas très bien de ces objets « fantômes » ! De plus, les problèmes à résoudre pour atteindre des performances raisonnables nécessitent de représenter des connaissances typographiques complexes : taille et inclinaison des hampes, taille des mesures, affichage des polyphonies, gestion des multiples portées simultanées, problèmes de justification et de rupture de page, etc. Enfin, les partitions réalistes comportent plusieurs centaines

3. Il s'agit plus précisément d'une ellipse légèrement inclinée, dont le trait est plus plein sur les bords que sur les sommets.

de notes, ce qui exclut en pratique l'utilisation de ces techniques. Ici encore, si la structuration par objets permet de réduire considérablement la complexité, celle-ci n'est pas pour autant totalement vaincue et de nouveaux paradigmes de programmation ou de représentation de connaissances doivent être identifiés pour permettre la construction aisée de tels outils.

13.7 Autres applications

Bien d'autres activités musicales peuvent ainsi être modélisées par des objets et la place manque pour les décrire toutes. Voici quelques unes d'entre eux, le lecteur est invité à consulter les références pour plus de détails.

13.7.1 Classification de sons et programmation de synthétiseurs

Ce problème concerne la spécification de sons pour les synthétiseurs du commerce. Les synthétiseurs commercialisés aujourd'hui proposent des modes de synthèse sonore extrêmement sophistiqués (par modulation de fréquence, soustractive, par modèles physiques), que les musiciens ne maîtrisent généralement pas, ce qui rend leur programmation quasiment impossible. Or les experts en programmation de synthétiseurs utilisent des connaissances souvent explicites, de surface, c'est-à-dire relativement indépendantes des modèles de synthèse effectivement employés par l'appareil. Un exemple typique de transformation de son est « pour rendre un son plus gros, on peut le doubler par une copie conforme et désaccorder très légèrement sa copie ». Reconnaissant que la plupart des règles expertes concernent les transformations applicables aux sons, plutôt que la fabrication de sons à partir de zéro, l'étude décrite dans [Rolland et Pachet, 1996] a conduit à construire une taxonomie de sons fondée sur les transformations applicables. L'idée est alors de classer un son de départ dans cette taxonomie, afin d'en déduire une liste de transformations, associée à chaque type de son trouvé par le classifieur. Cette liste est ensuite proposée à l'utilisateur, qui en sélectionne une. Puis la transformation est appliquée, conduisant alors à un nouveau son, qui est à son tour classé, puis le cycle recommence.

Du point de vue des représentations mises en jeu, ce système présente l'originalité d'être bicéphale : la partie classificatoire est réalisée dans le formalisme des logiques de descriptions (le système BACK, voir *chapitre 11*). Le résultat de cette classification est une liste de types (les classes de son, par exemple `BrassyAble`, `DecayAble`), auxquels sont associés des *noms* de transformations. Le nom de transformation une fois choisi par l'utilisateur, la transformation elle-même est effectuée par une méthode SMALLTALK ayant ce nom comme sélecteur. Les deux parties coopèrent pour réaliser le cycle de base, avec choix de la transformation par l'utilisateur à chaque cycle (voir figure 13.7).

13.7.2 Simulation d'improvisation

L'improvisation est une des activités humaines les plus fascinantes qui soit. Des travaux en cours tentent de modéliser cette activité, en l'envisageant comme

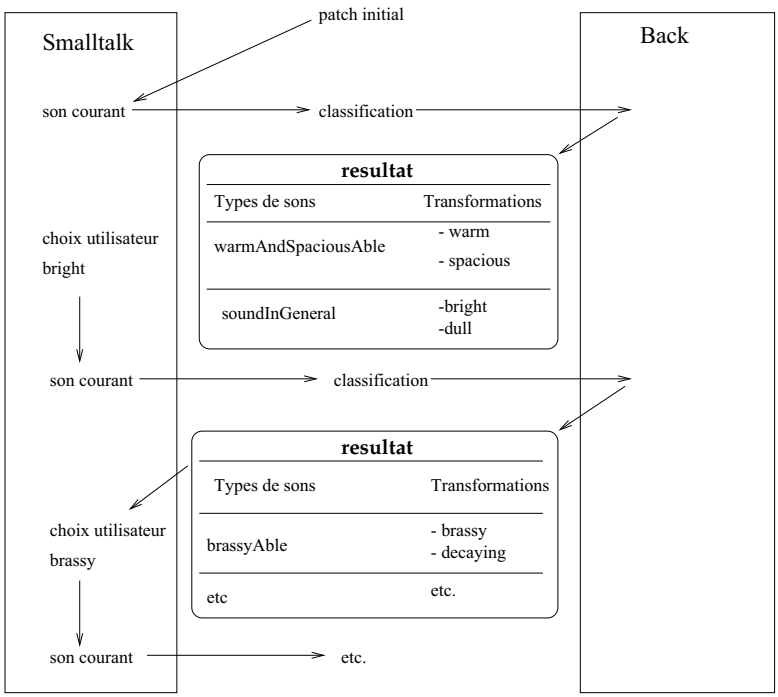

FIG. 13.7: *Le schéma de raisonnement pour la classification de patches de synthétiseurs. La partie classification est réalisée en* BACK*, l'application de la transformation sélectionnée en* SMALLTALK

une forme particulière de résolution de problèmes faisant intervenir une boucle de contrôle dirigée par le « temps restant disponible », et exploitant une représentation des actions musicales par objets [Ramalho et Pachet, 1994], ainsi qu'une mémoire musicale sous forme de cas [Ramalho et Ganascia, 1994]. Ces actions musicales et les cas sont représentés par des objets temporels particuliers et utilisent la représentation des objets temporels vue à la section 3. Ce modèle, qui donne lieu à une mise en œuvre aisée, permet d'étudier précisément ce qui, dans l'activité d'improvisation, relève de la simple juxtaposition de *patterns*, de ce qui relèverait d'une forme de « créativité ».

Un autre travail en cours dans le même domaine concerne le problème de la caractérisation des styles musicaux. Les techniques de programmation dynamique sont utilisées pour détecter des motifs récurrents, dans de larges corpus de mélodies : les improvisations de Charlie Parker [Rolland et Ganascia, 1996], et constituent un autre exemple d'utilisation de la bibliothèque de classes MUSES.

13.8 Conclusion

Nous avons montré quelques exemples typiques de problèmes musicaux nécessitant de représenter divers types de connaissances musicales, pour lesquelles la programmation par objets apporte des réponses intéressantes. Les mécanismes de la programmation par objets permettent ainsi de représenter une couche de base substantielle qui sert de point de départ pour l'élaboration de diverses applications. Mais ces mécanismes ne suffisent pas à exprimer bon nombre de connaissances complexes : on fait alors appel à des ontologies spécialisées, aux contraintes, aux règles, à la classification, à la programmation dynamique, etc. pour enrichir le pouvoir d'expression des objets de base, voire construire de nouveaux langages. Certains problèmes restent cependant difficiles à résoudre et mettent en évidence des complexités non encore maîtrisées. Ces problèmes musicaux soulèvent, comme nous l'avons vu, des questions fondamentales de représentation de connaissances, qui mettent à l' épreuve de manière fructueuse les paradigmes actuels de programmation et de représentation.