REPRÉSENTATION DE CONNAISSANCES SUR LA PROGRAMMATION DE SYNTHÉTISEURS

P.Y. Rolland, F. Pachet, LAFORIA-IBP, Université Paris 6

E-mail: rolland/pachet@laforia.ibp.fr

Résumé

Le problème de l'interface homme-machine est aujourd'hui critique dans le domaine des synthétiseurs du commerce. Nous nous attachons à capturer la couche superficielle de connaissances—le savoir-faire des "experts en programmation de sons (ou patches)". Nous proposons un cadre conceptuel et fonctionnel pour représenter ce savoir superficiel dans le but de fournir une aide aux musiciens non-experts : leur permettre de créer des patches de synthétiseur du commerce de manière plus intuitive qu'avec les interfaces actuelles. L'idée centrale de notre architecture est celle d'une classification des sons en fonctions des transformations que l'expert sait leur appliquer. Notre cadre conceptuel est fondé sur une représentation des sons à deux niveaux, combinant programmation par objets (Smalltalk) et mécanismes de classification (BACK). Nous proposons un schéma d'intégration des deux paradigmes, et illustrons notre approche par un système prototype d'aide à la programmation pour synthétiseurs de la famille Korg 0xx/W.

1. De la synthèse assistée à la programmation assistée

Métaphore de l'unité de soins intensifs

La remarque suivante est à l'origine de notre approche. Dans une unité de soins intensifs, une infirmière sait en général parfaitement se servir d'un respirateur artificiel. Néanmoins celle-ci peut être "experte" en matière de manipulation de ces appareils sans pour autant posséder une connaissance théorique de leur mode d'action, ou des aspects médicaux liés au diagnostic, au traitement etc. Il se peut qu'elle soit plus experte que le médecin; mais elle possède uniquement le savoir dont elle a besoin, un savoir superficiel lui indiquant comment utiliser l'appareil avec efficacité et sécurité. Par analogie, nous comparons d'un côté les experts en *programmation de patches* qui savent programmer efficacement un synthétiseur, de l'autre les experts en *synthèse sonore* pour qui les transformées de Fourier et le traitement de signal n'ont aucun secret mais qui ne savent pas comment, par exemple, rendre un son "plus chaud" sur un synthétiseur "hardware" du commerce tel que Yamaha W-7, E-Mu Proteus ou Korg 05R/W.

Nous visons à représenter les connaissances manipulées par la première catégorie d'experts, en affirmant qu'une telle représentation est possible et fournit le moyen d'enrichir les interfaces actuelles de programmation de synthétiseurs.

Paradigme de la Synthèse Assistée par Ordinateur (S.A.O.)

L'observation qui précède est à la base de notre approche en matière de programmation de synthétiseurs assistée par ordinateur (P.S.A.O.). De notre point de vue, les objectifs principaux d'un système de S.A.O. sont 1) d'aider le musicien à créer des sons, et 2) de l'aider à comprendre une technique de synthèse. Un certain nombre de travaux ont été consacrés à la conception de tels systèmes. Ceux ci se fondent sur des paradigmes variés, allant de la *synthèse par règles* implémentée par le système CHANT (Rodet et al. 1984) à *l'assistance adaptative à la synthèse* (ARTIST, de Miranda (1992)). Parmi les systèmes existants on peut également citer L'Intuitive Sound Editing Environment de Vertegaal & Bonis (1994), Kyma/Platypus (Scaletti 89), Javelina (Hebel 89), DMIX (Openheim 89) etc.

Conséquences

L'étude des système de SAO existants montre que :

1) la majorité des travaux ont concerné, non pas des synthétiseurs du commerce, mais des synthétiseurs informatiques sophistiqués qu'utilisent très peu de musiciens.

2) La plupart des systèmes proposent des environnements dans lesquels le musicien définit et manipule des objets et structures sonores. A notre connaissance, aucun de ces systèmes ne se propose de représenter directement les connaissances pratiques des programmeurs de patches experts.

C'est sur ces constatations que s'appuie notre choix d'aborder la *PSAO*, plutôt que la PAO proprement dite. Notre approche se fonde sur deux hypothèses:.

2. Hypothèses

H1. Les programmeurs de patches experts font appel à des connaissances superficielles

Nous faisons l'hypothèse que la programmation de synthétiseurs du commerce nécessite beaucoup de savoir faire et peu de connaissances "fondamentales". Le cas de la synthèse FM est éloquent à cet égard. L'histoire à succès de la famille de synthétiseur DX montre que bien des patches "fameux" ont été créés par des programmeurs possédant un savoir "superficiel" acquis grâce à l'expérience et à l'observation de patches programmés par leurs pairs. Pour se convaincre du rôle limité joué par la compréhension des aspects théoriques complexes, on se référera aussi aux tentatives faites pour répandre la théorie FM (Chowning et Bristow 1986) où l'accent est clairement mis sur les aspects pratiques. La "règle de savoir-faire" (R) ci-dessous est typique de la programmation FM, et peut être utilisée par un programmeur sans que celui-ci en comprenne le fondement théorique.

(R) On peut obtenir un son "tremolo" (modulation d'amplitude) en fixant la fréquence porteuse à une valeur basse et en réglant la modulante à une valeur variable en fonction de la note jouée.

H2. Le "savoir expert" concerne surtout les transformations sur les sons

Il est bien connu que les programmeurs de patches experts préfèrent, pour obtenir un son donné, partir d'un patch proche de l'objectif et le *transformer* plutôt que de "partir de zéro". Ainsi, considérant que la majorité de l'expertise concerne les transformations, nous nous attachons à représenter les connaissances liées plutôt aux transformations qu'aux sons eux-mêmes. Nous présentons ici deux exemples typiques des *règles transformationnelles* que nous souhaitons prendre en compte dans notre système. La première est applicable à tout synthétiseur possédant un filtre passe-bas paramétrique. La seconde règle est elle aussi applicable à une majorité de synthétiseurs du commerce, mais n'est valide que pour des sons pseudo-harmoniques, c'est à dire dont on perçoit la hauteur (exemple: son de piano, par opposition à un son de cymbale) :

(R1) On sait rendre un son *plus brillant* en augmentant la fréquence de coupure du filtre.

(R2) On sait rendre un son *plus chaud* en dupliquant le son et en imposant un *léger désaccord* entre le son initial et sa copie.

3. Règles transformationnelles et classification des sons

Organisation des connaissances

Le genre de règles transformationnelles décrites ci-dessus se représentent aisément par l'intermédiaire d'un système à base de connaissances, c'est à dire par des règles de production. Bien sûr comme nous l'avons vu toutes les transformations ne s'appliquent pas à tous les types de sons. Plus précisément les transformations s'appliquent à des types de sons "origines" et donnent lieu à des types de sons "cibles". Ces derniers sont caractérisés par de simples adjectifs ("brillant, "chaud", etc.). Afin d'organiser ces types de sons, il faut faire appel à un mécanisme de classification . Dégager une classification concernant les types de sons "cibles", basée par exemple sur les résultats de la psychoacoustique (Wessel 1979, Grey 1975) serait inadéquat. En effet un son donné doit être classifié en fonction des *transformations qu'il peut subir* et non de celles qui ont permis de l'obtenir: par exemple, nous ne cherchons pas à ranger les sons "cuivrés" au sein d'une quelconque typologie; au contraire, nous visons à identifier les "sons que l'on sait rendre cuivrés" grâce à une transformation connue.

Dans cette conception, être instance d'un type de sons signifie simplement être capable de subir les différentes transformations associées à ce type de sons, ainsi les noms des types de sons n'ont pas de sens direct. Pour des raisons pratiques, nous donnons aux types de sons un nom arbitraire construit sur le préfixe 'pré', par exemple préCuivré ou préSourd. Dans le cas de types de sons pouvant subir plusieurs transformations spécifiques, nous utilisons un nom composé du type préBrillant-Cuivré. Chaque transformation est associée à un type de sons particulier : par exemple, la transformation (R2) est associée au type de sons préChaud, la transformation (R1) au type préBrillant-Cuivré.

En résumé, la tâche principale du système est de classifier les sons selon une classification pré-définie. La figure 1 présente une partie de la hiérarchie de types de sons pour le Korg 05R/W. Les types de sons sont matérialisés par des rectangles en dessous desquels sont indiqués les transformations associées. Les flèches traduisent des liens "type/sous-

type".

Paradigme de représentation

Compte tenu de la nature des connaissances à représenter concernant la classification nous avons intégré, au sein de notre cadre de représentation, à la fois 1) des mécanismes de classification afin de manipuler les hiérarchies de sons, et 2) un langage procédural permettant d'exprimer les transformations appliquées au son courant. Des tentatives ont été faites (Yelland 1992) pour intégrer la classification automatique dans les langage orientés objets (LOO) mais aucun système intégré n'est directement disponible. Compte tenu du compromis expressivité / efficacité / complétude auquel sont soumis les systèmes de classification existants (Heinson et al. 94), nous avons opté pour le système BACK (Hoppe et al. 1993), une implémentation du formalisme des logiques descriptives — cf. section 4.

Une logique descriptive pour classifier les sons

Les sons que nous modélisons ont cette particularité qu'ils peuvent se décrire par des symboles au lieu de simples tableaux de valeurs de paramètres. Par exemple un son du Korg 05R/W se compose d'une à seize parties ("parts"), chacune composée d'une à deux voix ("voices"), etc. En bout de chaîne, ces structures abstraites sont décrites en termes de valeurs *terminales* de paramètres ('temps d'attaque', 'nom de la forme d'onde' etc.). Cette forte structuration est bien adaptée aux techniques de *classification de termes* dans lesquelles l'utilisateur établit "déclarativement" les liens structuraux et le système se charge des tâches d'inférence (classification et subsomption).

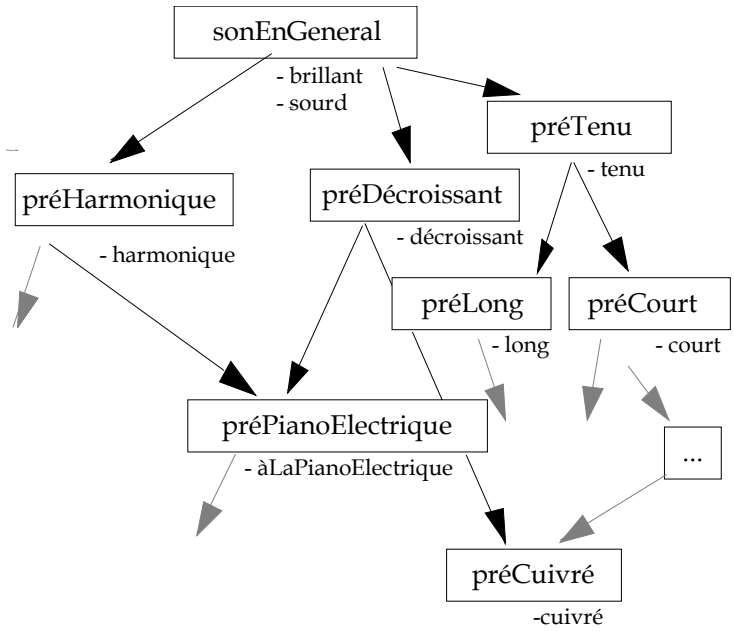


Figure 1. Hiérarchie de types de sons partielle pour le synthétiseur Korg 05R/W

Un LOO pour représenter les transformations

Les transformations, pour leur part, doivent faire l'objet d'une représentation effective et pas uniquement abstraite, ce que BACK ne permet pas directement. De plus, la programmation de l'interface utilisateur et de la communication MIDI est bien plus aisée avec un LOO (en l'occurrence Smalltalk-80). Les sons sont représentés comme *instances* de classes de sons, les transformations comme *méthodes* pour ces classes.

3) Un mécanisme de couplage entre les deux représentations

Faire coexister deux paradigmes de représentation n'est pas trivial, en particulier en raison de problèmes de cohérence : les instances d'un même type de sons sont de nature complètement différente suivant que celui-ci est représenté comme une classe (LOO) ou un concept (LD). La section 6 décrit précisément le mécanisme de couplage que nous proposons.

4. La classification automatique des sons

Les logiques de description (LD) et BACK

Issues de travaux de recherche initiés vers 1980, les LD sont des formalismes de représentation proposant une organisation hiérarchisée des connaissances où tout *concept* hérite les propriétés des concepts *plus généraux* que lui et entretient avec les autres concepts des liens binaires appelés *rôles*. Les nombreuses implémentations existantes (Heinsohn et al. 1994), dérivées du précurseur KL-ONE (Brachman 1985), offrent différents "services" d'inférence comme les tests de subsomption et les requêtes de classification.

Nous avons utilisé l'implémentation BACK (Berlin Advanced Computational Knowledge, version 5 basée sur Prolog) que nous décrivons ici dans ces grandes lignes. Les concepts sont des ensembles d'*objets* introduits intensionnellement ou extensionnellement, et les objets sont instances d'un ou plusieurs concepts. Les concepts peuvent être primitifs ou définis, selon qu'ils donnent lieu à des conditions nécessaires (resp. nécessaires et suffisantes) de classification pour leurs instances. Le *contenu d'un rôle* (*role-filler*) correspond au second membre de la relation binaire correspondant à ce rôle. Toutes ces notions quelque peu abstraites seront illustrées par divers exemples ci-après.

L'information est introduite dans BACK à travers la création de *termes* (concepts ou rôles) et d'objets, ou à travers l'usage de *règles non définitionnelles* imposant des relations logiques particulières entre les concepts. Les services d'inférences comprennent la *classification* qui fournit la liste des concepts instanciés par un quelconque objet, et le *test de subsomption* qui indique si un concept c1 est plus spécifique que c2, c'est à dire si toute instance de c1 est instance de c2.

Description "technique" des sons

Dans la mémoire du Korg 05R/W, un son est représenté par une liste de valeurs de paramètres de synthèse. La *description technique* des sons correspond à cette représentation de bas niveau. La figure 2 montre l'architecture de la synthèse telle que la présente Korg. On retrouve les *parties* et les *voix* introduites plus haut. Sur le schéma de la figure 2, les cases rectangulaires symbolisent un premier type d'unités chargées de générer ou de traiter le signal audionumérique (Oscillator, Variable Digital Filter ou VDF, processeur d'effets, etc.). Les cases arrondies symbolisent, elles, un second type d'unités : des générateurs d'enveloppe ou de modulation contrôlant l'évolution temporelle des paramètres des unités de type 1 (exemple: pitch enveloppe generator, VDF modulation generator). Plusieurs unités sont associées au sein d'une structure de type 'voix'. En ce qui concerne les parties, nous avons dit que celles-ci regroupent chacune une ou deux voix. Les deux voix d'une partie à deux voix ont certaines de leurs unités en commun (pitch enveloppe generator, VDF modulation generator). En revanche, une partie ne comportant qu'une voix se résume en gros à cette voix.

Représentation des sons dans BACK

La représentation BACK traduisant le diagramme de la figure 2 se fait par l'introduction de deux types de concepts: concepts fondamentaux et concepts abstraits.

Concepts fondamentaux

Les *termes* constituant la représentation 'technique' des sons seront appelés "fondamentaux" car ils sont une transcription presque directe de l'architecture du modèle de synthèse. Par exemple, le concept **voix** ne peut être introduit à partir de concepts plus généraux que lui, autres que le concept prédéfini **anything.**Les conditions de classification d'objets associées à **voix** étant seulement nécessaires, nous l'introduisons comme un concept *primitif* (d'où le symbole BACK :<).

```
voix :< anything.
```

Le concept forme_dOnde est *défini* (symbole :=), car son introduction en extension se traduit par des conditions *nécessaires et suffisantes* de classification d'objets. Pour des raisons évidentes, nous avons considérablement écourté ici la liste des 300 formes d'ondes stockées dans la mémoire du 05R/W.

```
forme_dOnde := attribute_domain([sinus, carré, triangle, bruitBlanc]).
```

Le rôle primitif **aForme_dOnde** peut alors être introduit pour spécifier la forme d'onde sur laquelle est basée une voix. Nous qualifions un tel rôle de *terminal*, signifiant que son contenu est directement lié à des valeurs de paramètres (ici par exemple: 'sinus', 'carré' etc.) et non à des structures abstraites, comme c'est le cas pour le rôle

aVoix utilisé plus bas.

```
aForme_dOnde :< domain(voix) and range(forme_dOnde).
```

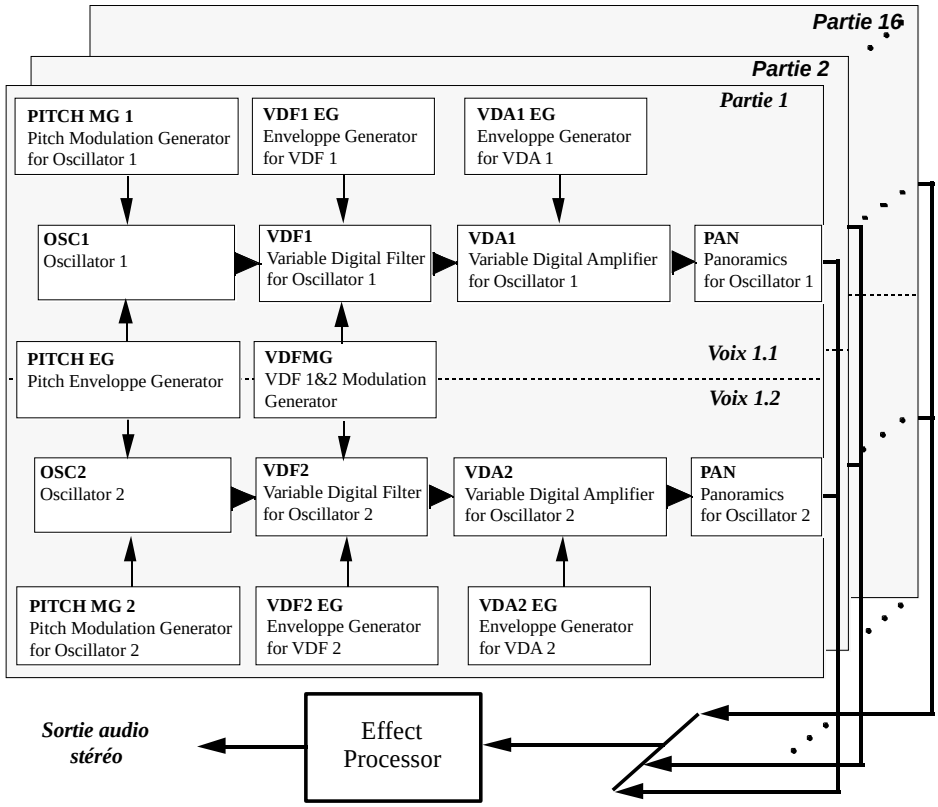


Figure 2. Représentation "technique" des sons : schéma de synthèse du 05R/W

Un autre concept défini est **partieBiVoix**, dérivé du concept primitif **partie** par une restriction sur la cardinalité du rôle **aVoix** :

```
partieBiVoix := partie and exactly(2, aVoix).
```

Concepts abstraits

Au dessus de cette première couche de représentation, nous construisons une hiérarchie de concepts représentant la partie "structurale" de l'expertise. Comme on le verra plus loin, la définition des transformations fait intervenir ces concepts . Nous répartissons ces derniers en trois catégories :

i) des concepts abstraits construits à partir de concepts fondamentaux

Parmi ces concepts, des descriptions partielles de sons comme enveloppeTenue, définie comme une enveloppe dont la valeur du paramètre 'niveauDeSustain' est non nulle. En termes de synthèse, cela permet de décrire des sons dont, par exemple l'amplitude, se stabilise à une valeur non nulle (oscillation entretenue).

```
enveloppe                    and the(niveauDeSustain, ge(1))
                             => enveloppeTenue.
```

De façon similaire sont introduits les concepts abstraits jouant un rôle dans la tâche de représentation du type de sons préCuivré (voir plus bas). Par souci de clarté, dans les exemples qui suivent nous utilisons des valeurs numériques explicites (par exemple dans 'the(tempsAttaque, ge(17) and le(25))') tandis que dans notre implémentation nous avons recours à des constantes nommées comme 'tempsAttaqueMinPourEnveloppeFiltreCuivrée"

```
enveloppeFiltre          and the(tempsAttaque, ge(17)
                         and le(25))
                         and the(niveauAttaque, ge(85))
                         and the(tempsDecroi, ge(60) and le(75))
                         and the(niveauInterm, ge(25)
                         and le(35))
                         and the(niveauTenu, ge(25)
                         and le(35))
                        => envFiltrePréCuivrée.

filtreVar                and the(aEnveloppe,
                        envFiltrePréCuivree)
                        => filtreVarPréCuivré.

voixBrillante            and the(aFiltre, filtreVarPréCuivré)
                        and the(aAmpli, ampliPréCuivré)
                        => voixPréCuivrée.

partie                   and atleast (1, aVoix, voixPréCuivrée)
                        =>partiePréCuivrée.
```

D'autres exemples de concepts abstraits reflétant les connaissances expertes structurelles sont ceux décrivant des sons "non transformables". Contrairement aux précédents, ces concepts abstraits fournissent des descriptions complètes de sons pour lesquels on ne connaît pas de transformation particulière mais qui servent à décrire des sons transformables :

```
son                      and no(aPartie, partieAvecVoixInharmonique)
                         => SonHarmonique
```

Il est à noter que dans la mesure où tout son est subsumé par le type de son transformable 'sonEnGénéral', même les instances de sons non transformables sont passibles de transformations comme 'rendre brillant' ou 'rendre sourd'.

ii) des concepts décrivant des sons transformables

Voici quelques exemples de tels concepts :

```
SonHarmonique            and some (aPartie, partiePréCuivrée)
                         => sonPréCuivré

son                      and all(aPartie, partiePréTenue)
                         => sonPréTenu.

son                      => sonEnGénéral.
```

iii) des concepts représentant les transformations elles-mêmes :

A chaque type de son nous associons une liste de transformations représentées comme des simples chaînes de caractères. Cette liste est matérialisée par un sous-concept de **transformationsPossibles**, lequel liste l'ensemble des transformations pour l'ensemble des types de sons

```
transformationsPossibles:= attribute_domain ([chaud, cuivré, sourd,
décroissant (...)]).

supporteTransformation :< domain(son) and range
(transformationsPossibles).

sonPréCuivré :< supporteTransformation : cuivré.

sonPréBrillant-Cuivré :< supporteTransformation : brillant and cuivré.
```

5. Représentation des sons par objets

Une telle représentation des sons est tout à fait naturelle dans notre contexte où l'accent est mis sur les transformations. Chaque transformation est représentée par une méthode Smalltalk définie dans la classe **sonCourant**. Cette méthode modifie la valeur des paramètres 'terminaux' par le biais d'un ensemble pré-défini de "modifieurs". Voici par exemple la méthode rendant un son "tenu".

```
soisTenu
     voix do: [:v | v sois: #tenue].
     ...
```

où **sois:** est définie comme suit

```
sois: unSymbole
     unSymbole = #tenue ifTrue:
     [enveloppeFiltre tempsDécroissance: 99.
     enveloppeAmpli tempsDécroissance: 99].
     unSymbole = #...  ifTrue: [...]
```

6. Intégration

Le schéma d'intégration est fondé sur deux principes :

Concepts

Chaque concept fondamental est représenté par une classe Smalltalk. Chacun des rôles du concept est représenté par une variable d'instance de la classe correspondante. Bien entendu, la représentation en Smalltalk est rudimentaire comparée à son analogue dans BACK, puisque les types et les cardinalités ne sont pas pris en charge. Les sons eux-mêmes sont représentés par des instances de la classe **sonCourant**, leurs paramètres par des instances des classes correspondantes.

Transformations

Celles-ci sont représentées par des méthodes associées à la classe **sonCourant.** Chaque méthode modifie le son courant en changeant certaines de ses valeurs de paramètres. Ainsi, la sémantique du symbole BACK représentant les transformations est fournie par la méthode Smalltalk correspondante. La figure 3 illustre le cadre de représentation à deux niveaux issu de ce schéma d'intégration.

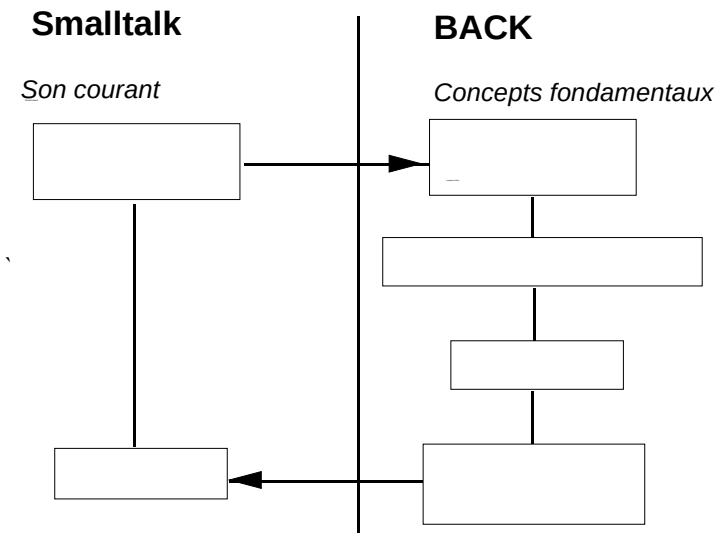


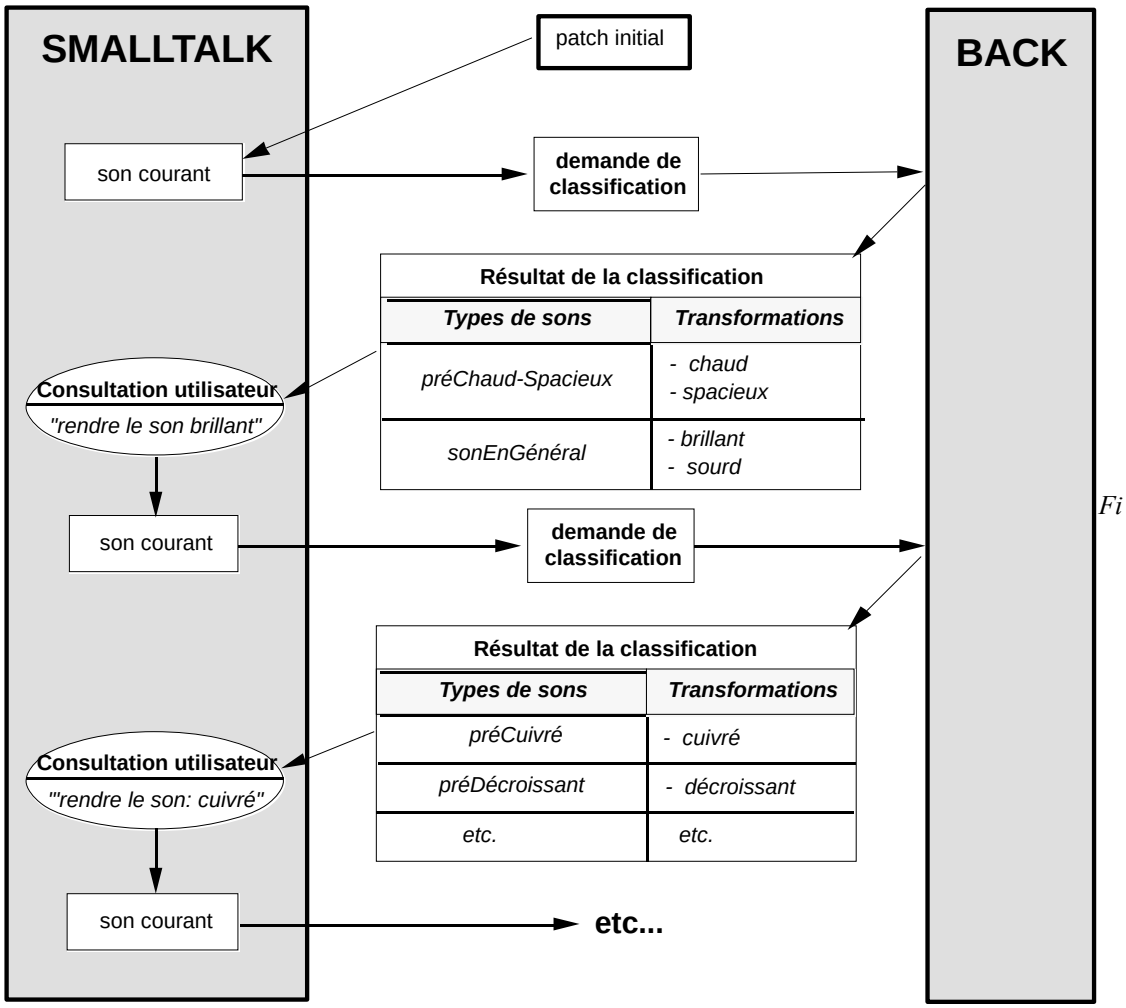
Figure 3. Les deux représentations des sons et leur connexion

7. Exécution

Une séance de travail avec notre système peut être vu comme un cycle itératif d'opérations :

1. Tout d'abord, l'utilisateur choisit un patch initial.
2. Le patch est transmis à BACK en vue d'être classifié. BACK rend la liste des types de sons instanciés, avec la liste des transformations associées. Ces données sont transmises à Smalltalk.
3. L'utilisateur choisit une des transformations proposées. Les paramètres de l'objet Smalltalk sonCourant, ainsi que ceux du synthétiseur réel sont modifiés en conséquence, et l'utilisateur a la possibilité d'écouter et de jouer de ce nouveau son.
4. Retour à l'étape 2.

Le figure 4 schématise une séance utilisateur typique.



gure 4. Exemple de séance utilisateur.

8. Ajout d'un nouveau type de sons

Au stade actuel de notre prototype, l'ajout d'un nouveau type de sons n'est pas une tâche aisée. Cela exige d'apporter trois modifications au système :

1. Ajouter un ensemble de termes décrivant le nouveau type de sons en fonction des types existants et des concepts fondamentaux. Pour illustrer notre propos, supposons que l'on souhaite ajouter le type de sons préChoral-Bruité. On serait amené à introduire les termes et règles suivants :

```
ondePourNappe              := aset([orgue, fluteBoucle, sinus, square],
                           forme_dOnde).

voix                       and voixTenue
                           and the(aForme_dOnde, ondePourNappe)
                           => voixPourNappe.

(...)

sonDoux                    and sonNappé
                           and atLeast(1, aPartie, partieBiVoix)
                           => sonPréChoral-Bruité
```

2. Spécifier, dans BACK, quelles transformations spécifiques sont applicables au nouveau type de sons.

```
sonPréChoral-Bruité         => supporteTransformation:
                            choral and bruité.
```

3. Spécifier les transformations elles-mêmes côté Smalltalk. Par exemple :

```
soisChoral
     self parties désaccordEnDemiTons: 0.05; soisSymétrique
     self adopteVibratoDeType: #Choral.
```

où **désaccordEnDemiTons**:, **soisSymétrique** et **adopteVibratoDeType**: sont des méthodes soit définies précédemment soit rajoutées pour l'occasion, par exemple :

```
désaccordEnDemiTons: unNombre
     self voix1 accordFin: (0.5 * unNombre) negated.
     self voix2 accordFin: (0.5 * unNombre).
```

Dans une version future il est prévu de doter le système d'une capacité d'apprentissage (basée probablement sur l'apprentissage inductif à partir d'exemples), ce qui autorisera la définition de nouveaux types de sons par l'intermédiaire de patches représentatifs simplement présentés au système. De plus chaque transformation pourra être définie à partir d'un ensemble de paires de patches, chacune regroupant un exemple de son d'origine et ce même son après transformation.

9. Conclusion

La contribution principale de notre travail de recherche concerne l'implémentation d'un formalisme de représentation sophistiqué, celui des logiques descriptives, en vue de capturer les connaissances superficielles se rapportant à la programmation de synthétiseurs. L'exploitation de ces connaissances permet de proposer au musicien de naviguer dans l'espace des timbres d'un synthétiseur du commerce de façon intuitive, réduisant ainsi la complexité de l'acte de création de sons.

Nous avons conçu et testé sur le synthétiseur Korg 05R/W un prototype. Ce travail est en cours, et nos efforts se concentrent actuellement sur : 1) le raffinement de l'interface utilisateur, 2) l'amélioration de la communication entre BACK et Smalltalk, et 3) l'expérimentation avec des utilisateurs novices.

Références

Brachman, R.J., and Schmolze, J.G., 1985. An overview of the KL-ONE knowledge representation system. Cognitive Science, 9 (2). p. 171-216.

Chowning, J. Bristow, D. 1986. FM Theory & applications by musicians for musicians. Yamaha Music foundation, cop. 1986.

Goldberg, A., Robson, D. 1983. Smalltalk-80 : The langage and its implementation. Addison Wesley.

Grey, J. 1975. An Exploration of Musical Timbre. PhD dissertation, Stanford University Psychology Dept. CCRMA Report STAN-M-2.

Hebel, K, 1989. Javelina: An Environment for Digital Signal Processing Software Development. Computer Music Journal, Vol. 13, N. 2, Summer 1989. Réimprimé dans the Well-Tempered Object, S. Pope Ed. MIT Press, 1991.

Heinsohn, J. Kudenko, D. Nebel, B. Profitlich, H-J. 1994. *"A*n empirical analysis of terminological representation systems" , *Artificial Intelligence 68 (August 1994),* Elsevier, Allemagne, vol.2, p. 367-397.

Hoppe, T. Kindermann, C. Quantz, J. Schmiedel, A. Fischer, M. 1993. Back V5 *Tutorial&Manual*, Institut fûr Software und theoretische Informatik, W-1000 Berlin 10, Germany, march 1993.

KORG. Non daté. Korg 05R/W *Manuel d'utilisation*, AI^2 SYNTHESIS MODULE. Korg Inc.

Michalski, R. S. 1983. A theory and methodology of inductive learning. in Machine Learning: an Artificial Intelligence approach. Michalski, R.S., Carbonell, J.G. and Mitchell, T.M., eds. Palo Alto: TIOGA Publ. Co. California. pp. 83-134.

Miranda, E. 1992. From symbols to sounds : An AI-based investigation of Sound Synthesis (PhD Thesis Proposal). DAI Discussion Paper No. 117, Dept. of AI, University of Edinburgh.

Openheim, D. V. 1989. DMIX: An Environment for Composition. Actes de ICMC'89, Columbus, Ohio. International Computer Music Association.

Openheim, D. V. 1991. Shadow: An Object-Oriented Performance System for the DMIX Environment. Actes de ICMC'91, Montréal, Canada, pp. 281-284.

Rodet, X., Potard, Y., Barrière, J.B. 1984. The CHANT project : from the synthesis of the singing voice to synthesis in general. In The Music Machine, C. Roads (ed.), MIT Press.

Royer, V. Volle, P. 1994. "Introduction aux Logiques de Description", *Tutoriel RFIA'94*, 11 Janvier 1994.

Scaletti, C. 1989. The Kyma/Platypus Computer Music Workstation, CMJ, 13:2, Summer 1989. Réimprimé dans the Well-Tempered Object, S. Pope Ed. MIT Press, 1991.

Smith J.O. 1992. Physical Modeling Using Digital Waveguides. Computer Music Journal. 16:4, pp. 74-87. MIT Press.

Vertegaal, R., Bonis, E. 1994. ISEE : an intuitive sound editing environment. Computer Music Journal 18:2. MIT Press

Wessel, D. 1979. Timbre Space as a Musical Control Structure. Computer Music Journal. 3:2. MIT Press

Yelland, 1992. Experimental Classification Facilities for Smalltalk. Actes de OOPSLA' 92, Vancouver. pp. 235- 246.