Représentation de connaissances et langages à objets

Habilitation à diriger des recherches

Mémoire de synthèse

François PACHET

Résumé : Ce rapport contient le document présenté pour obtenir l'habilitation à diriger des recherches, soutenue le 15 mai 1997 à l'université Pierre et Marie Curie. Il fait état de recherches menées au laboratoire Laforia/LIP6, concernant l'exploitation des techniques de programmation par objets pour la conception et le développement d'applications en intelligence artificielle.

Abstract: This report is the document submitted to obtain the « Habilitation à diriger des recherches », defended May 15th at University Pierre et Marie Curie. The research described concerns the study of object-oriented programming for the design and development of Artificial Intelligence applications.

TABLE DES MATIÈRES

1. Introduction

Ce rapport synthétise mes activités de recherche de 1992 à 1997. Ces travaux sont une contribution au rapprochement entre génie logiciel et intelligence artificielle. Ils concernent plus spécifiquement l'utilisation des langages à objets pour la représentation de connaissances et la modélisation du raisonnement.

1.1. Génie logiciel et intelligence artificielle

Le génie logiciel a comme ambition de mettre l'art de l'Ingénieur au service de la construction rationnelle de programmes. Le génie logiciel s'est traditionnellement concentré sur les aspects formels et "bien posés" du développement de programmes (preuve, compilation et algorithmique traditionnelle), en faisant abstraction autant que possible des facteurs humains. Les exigences des utilisateurs d'aujourd'hui sont telles que le génie logiciel doit maintenant résoudre des problèmes beaucoup moins bien posés, depuis la spécification (changeante, floue, imprécise) des besoins de l'utilisateur, à la modélisation de domaines complexes.

A l'inverse, l'intelligence artificielle se nourrit, par essence, de problèmes mal posés, vagues, pour lesquels on ne connaît pas de solution algorithmique. Une de ses ambitions premières est de trouver des cadres conceptuels et informatiques pour représenter des connaissances, et les mettre en machine pour résoudre de tels problèmes. Un des freins au développement de l'IA provient sans doute du fait que cette discipline a jusqu'à présent ignoré les acquis et résultats en génie logiciel (Balzer, 1990). Un programme d'IA est ainsi le plus souvent un programme fermé, inextensible, utilisant des techniques et des langages non standards. Or aujourd'hui, les systèmes d'IA doivent être mis en œuvre dans des cadres technologiques de plus en plus ouverts et complexes.

La convergence entre IA et génie logiciel est donc naturelle. D'une part le génie logiciel s'occupe de plus en plus de problèmes relevant traditionnellement de l'IA. D'autre part l'IA doit s'adapter aux contraintes technologiques modernes. Pour construire les programmes de demain, il faut effectivement à la fois une technologie de plus en plus sûre, et de plus de plus en plus complexe, et des représentations de grandes quantités de connaissances, nécessaires à la fois à la bonne marche des programmes, et à leur adéquation aux besoins modernes. Cependant, la cohabitation de ces deux branches de l'informatique - intelligence artificielle et génie logiciel - n'est pas encore réalisée. Mes travaux de recherche sont une contribution à ce rapprochement, et s'inscrivent dans le contexte des langages à objets, qui ont pris leur source dans ces deux domaines (représentation des connaissances et génie logiciel), et y sont devenus omniprésents.

1.2. Curriculum

J'ai effectué mon stage de DEA en 1987 à l'IRCAM (département pédagogie), sous la direction de David Wessel et Jean-Louis Laurière. L'objectif du stage était de montrer que l'on pouvait utiliser des systèmes d'IA pour modéliser certaines tâches liées à la musique improvisée. J'ai identifié et étudié deux problèmes traitant de la musique improvisée : l'analyse harmonique de grilles d'accords, et la génération automatique d'accompagnement de Jazz, et réalisé deux systèmes qui ont nourri plusieurs de mes travaux par la suite. J'ai ensuite passé un an au titre de la coopération comme professeur d'intelligence artificielle à *l'Universiti Malaya*, Kuala-Lumpur (Malaisie), où j'ai continué mes travaux commencés en DEA et enseigné l'intelligence artificielle (1987-1988). J'ai ensuite fait une thèse au Laforia (Paris 6) sous la direction de Jean-François Perrot (1988-1992), dans laquelle je me suis intéressé à l'intégration de mécanismes d'inférence en chaînage avant dans les langages à objets. J'ai ensuite passé un an de stage post-doctoral au laboratoire LARC de l'Université du Québec à Montréal (bourse INRIA), dans l'équipe d'Hafedh Mili, où j'ai travaillé sur l'application du système *Cyc* à la génération de textes. J'ai eu par la suite l'occasion de collaborer à plusieurs reprises avec le laboratoire LICEF de la Télé-Université (Montréal), où j'ai travaillé sur les systèmes conseillers et participé à l'élaboration de la notion de *système épiphyte*, avec Sylvain Giroux et Gilbert Paquette, ainsi qu'avec le laboratoire Héron de l'Université de Montréal où j'ai été invité pour travailler sur des problèmes analogues, dans un contexte industriel. Je suis maître de conférences au Laforia depuis 1993, où je poursuis mes recherches avec des étudiants de troisième cycle.

1.3. Structure du document

Ce document commence par situer mes travaux par rapport aux recherches sur les langages à objets (section 2). Les 4 sections suivantes décrivent les recherches menées sur des points particuliers : objets et règles autour du système *NéOpus* (section 3), systèmes conseillers et l'architecture *Epitalk* (section 4), objets et satisfaction de contraintes à travers le système *BackTalk* (section 5), et enfin représentation du sens commun (section 6). La section 7 est consacrée à mes travaux en informatique musicale, et décrit plusieurs systèmes musicaux développés autour de la bibliothèque de classes *MusES*. La dernière section propose des directions de recherche à court et moyen terme.

2. Langages à objets et représentation de connaissances

Avant de décrire mes activités concernant la représentation de connaissances et les langages pour ce faire, je voudrais situer ma démarche par rapport aux travaux sur les langages à objets en général.

2.1. Deux familles de langages à objets

Les langages à objets sont devenus incontournables dans toutes les branches de l'informatique. En programmation, les mécanismes de base sont maintenant bien connus et maîtrisés (voir par exemple (Perrot, 1995)). En intelligence artificielle, les langages à objets sont aussi devenus omniprésents, bien que le terme « objet » revête des significations bien différentes suivant les communautés. On distingue traditionnellement deux familles de langages à objets[1] : les langages de programmation par objets et les langages centrés-objets.

Les langages de programmation par objets (PPO), comme Smalltalk, Eiffel, C++, ou Java, reposent sur la notion d'objet comme entité regroupant une *structure* et un *comportement*, tous deux définis dans la *classe* de l'objet, et donc communs à toutes ses instances. La structure représente le « c'est fait comment » d'un objet. Pratiquement cette structure permet d'organiser les objets en un réseau dont les nœuds sont les objets et les liens sont les relations entre objets. Le comportement est l'aspect « sait faire quoi », et est représenté par une batterie de procédures, appelées méthodes, qui sont déclenchées par envoi de messages explicites. Enfin, les classes sont organisées en une hiérarchie d'héritage permettant la définition par addition de structure et de comportement. La combinaison de ces trois principes simples (agrégation, instanciation et héritage) est détonante, et a fini par s'imposer dans toutes les branches de l'informatique. Les langages de programmation par objets ont été beaucoup étudiés du point de vue de leurs propriétés, de leur implémentation et de leurs aspects réflexifs et concurrents, notamment en France avec les travaux de Pierre Cointe (Cointe, 1984), Jean-Pierre Briot (Briot, 1989b) et Jacques Ferber (Ferber, 1989). Un certain nombre de travaux sur l'utilisation de ces langages pour résoudre divers problèmes d'intelligence artificielle ont été ensuite menés sur ces bases : Robert Voyer sur l'implémentation efficace de mécanismes d'inférence (Voyer, 1989), Francis Wolinski sur la représentation des hiérarchies de parties et des multi-facettes (Wolinski, 1990), Philippe Krief sur la représentation de mécanismes d'évaluation (Krief, 1990), ou Robert Bourgeois sur la connotation et la coréférence (Bourgeois, 1990), pour ne citer que les plus parisiens d'entre eux.

Les langages dits « centrés objets », ou aussi de « représentation par objets » (RPO) reposent sur des structures de *frames*, dans la lignée du papier fondateur de Minsky (Minsky, 1975). Le Système *Shirka* est un des représentants de cette famille en France (Euzenat & Rechenmann, 1995). De nombreux travaux de recherche ont été poursuivis prenant ces langages comme base de représentation pour étudier des problèmes similaires à ceux mentionnés ci-dessus : représentation des hiérarchies de parties et des multi-facettes (Marinõ, 1993), des tâches (Willamowski, 1994),

[1] En excluant la famille des langages à prototype.

(Pierret-Gobreich, 1996), des règles de production[2] (Euzenat & Rechenmann, 1995), des contraintes (Kökeny, 1994), etc.

Le parallélisme frappant entre les thèmes de recherches pour ces deux familles de langages montre à quel point les questions abordées sont fondamentales et incontournables. Outre les différences de détail (valeurs par défaut, réflexes) les différences principales entre les deux familles de langages sont de deux ordres : le statut donné aux procédures, et le statut des classes. Précisons notre position sur ces deux points.

1. Statut des procédures

Le problème du statut des procédures est un problème fondamental, lié à la nature même de la notion de programme. Intrinsèquement liées à l'être dynamique de l'ordinateur, elles sont au cœur de la pensée informatique en général. Mais la procédure est, par définition, un mouvement, un passage entre états, et, si l'on sait bien parler et représenter des états comme des données, la représentation des procédures pose problème. On peut distinguer plusieurs tendances en fonction du point de vue adopté pour appréhender les procédures :

1) La description avec un méta-langage. En génie logiciel, on tente de trouver des cadres et des langages pour écrire les procédures avec le maximum de garanties, par exemple en proposant des mécanismes d'assertions pour spécifier des invariants ou des pré et post-conditions (Meyer, 1992). Ces assertions peuvent être vues comme une sorte de discours sur le programme, qui se tient dans un univers à part.

2) La suppression. Les puristes de la déclarativité considèrent les procédures comme des boîtes et des bêtes noires, qu'il s'agit de faire disparaître autant que possible. La plupart des langages de RPO adoptent un point de vue radical, qui consiste à supprimer purement et simplement les procédures du langage. Une autre solution est la déprocéduralisation au sens de Jacques Pitrat (Pitrat, 1990), qui consiste à utiliser des mécanismes d'amorçage (*bootstrap*) pour remplacer progressivement des procédures écrites manuellement par des procédures générées automatiquement à partir de connaissances déclaratives.

3) La réification. Une troisième manière de considérer les procédures est de les réifier, en tentant de les considérer comme des objets à part entière. C'est l'approche adoptée par exemple par (Willamowski, 1994) avec son système de tâches intégré au langage *Shirka*. D'autres réifications des actions sont possibles, en particulier celle que nous avons proposée pour les actions musicales (voir 7.2.2). C'est d'une manière générale cette dernière approche que nous avons suivie.

2. Statut des classes

Le deuxième point est celui du statut des classes. Les langages de PPO considèrent les classes comme des moules à objets. Les objets sont créés à partir de leur classe et ne peuvent en changer au cours de leur vie. Les langages de RPO définissent un lien plus flexible entre objet et classe, et mettent le mécanisme de classification au centre du langage. La classification prise comme mécanisme de base de représentation, les langages de programmation sont alors considérés comme inaptes à représenter, puisqu'un objet au sens de la PPO ne peut changer de classe, au

[2] Ces travaux ont été reniés par la suite par leurs auteurs, les règles étant considérées comme trop procédurales.

moins dans une vision naïve des choses. Il s'agit là d'une hypothèse forte sur la nature de la connaissance. Selon nous, la classification n'est qu'un mécanisme d'inférence parmi d'autres, et les problèmes que nous avons eu à résoudre ne se formulent pas tous dans ce cadre (en fait très peu d'entre eux passent le crible exigeant de ce formalisme). Dès lors il ne nous paraît pas naturel de le prendre comme point de départ. En revanche, pour les situations de représentations qui relèvent naturellement de ce paradigme, rien n'empêche de doter un langage de PPO d'un tel mécanisme (si-besoin !) comme cela a été fait et comme nous l'avons d'ailleurs expérimenté avec succès sur un système d'aide à la synthèse sonore (Cf. 7.7).

La littérature abonde de descriptions des points communs et des divergences, voire de tentatives d'unification, de ces deux familles de langages à objets (voir par exemple (Perrot & Napoli, 1996) ou (Carré et al., 1995)) ainsi que de formalisation de ces langages (Ducournau, 1996), notamment en prenant comme point de repère les logiques de description. S'agissant des problèmes de représentation difficiles, nous pensons que les langages de PPO sont ceux qui permettent aujourd'hui le plus facilement de construire des systèmes pour observer des phénomènes, sans imposer d'*a priori* sur la nature du système d'inférence.

2.2. Objets concrets, objets abstraits

L'opération de base qui justifie l'emploi des langages à objets pour la représentation de connaissances est la *réification*, opération mentale essentielle qui consiste à associer à un « quelque chose » du monde réel - un objet, un concept - un objet informatique (Ferber, 1989). Cette opération de réification est naturelle et sans problème pour la plupart des objets concrets, tangibles, solides, car ceux-ci relèvent le plus souvent d'un consensus. Mais l'expérience montre que les objets abstraits posent en revanche souvent problème, et la plupart des travaux en représentation de connaissances peuvent être interprétés comme des tentatives de « réifier l'abstrait ». Ces objets abstraits sont multiples et apparaissent dès que l'on cherche à modéliser des situations complexes : facettes et points de vues, systèmes et sous-systèmes, modèles et méta-modèles, règles et contraintes, heuristiques et conseils, objets liés à la représentation du temps, procédures et tâches, mais aussi types et sous-types, intervalles et plus généralement objets d'ordre supérieur. La difficulté n'est pas seulement que ces objets échappent au consensus (personne ne s'accorde aujourd'hui à donner un sens précis et unique au mot « contrainte » par exemple), mais aussi qu'ils obéissent à des lois particulières non directement assimilables par un langage à objets, quel que soit le dialecte considéré.

Mes travaux sont un ensemble de tentatives de réifier quelque uns de ces objets abstraits, sous la forme le plus souvent d'extensions de langages à objets de base.

2.3. De l'existence d'un langage de spécification

Derrière le débat entre programmation et représentation se situe un autre débat, portant sur l'existence de langages ou méthodes de spécification, indépendants de tout langage de programmation/représentation particulier.

En génie logiciel, certains extrémistes nient l'existence et l'intérêt de tels langages de spécification, et considèrent cette quête comme inutile et perdue d'avance, comme le défend, comiquement, Bertrand Meyer :

> « inaccurate as it would be to consider language issues only, the reverse mistake is just as damaging. A strange idea has become prevalent in some software circles: the claim that languages,

after all, are not that important. Requests are even heard here and there for a bizarre animal, the « language-independent methodology » - about as useful as a bird without wings, and just as likely » (Meyer, 1992), p. viii.

Dans les milieux moins fatalistes, on peut distinguer deux tendances vers l'abstraction : une tendance descendante qui part de spécifications abstraites, pour tenter ensuite de les concrétiser, et une tendance ascendante, qui part de l'existant pour l'analyser et tenter d'identifier des régularités utilisables a posteriori.

L'approche descendante est courante à la fois en génie logiciel et en intelligence artificielle. En génie logiciel, les travaux sur les langages de spécifications tentent d'abstraire les détails d'implémentation pour se concentrer sur les aspects formels et fonctionnels des programmes. En IA, de nombreux travaux reposent sur le postulat d'existence d'un niveau de connaissances (le *knowledge level*) indépendant d'un langage de représentation, introduit par Newel (Newell, 1982)). Ce postulat est à la source de nombreux travaux en acquisition des connaissances (le projet Européen KADS et ses successeurs), tentant aussi de fournir des cadres abstraits pour modéliser des systèmes à bases de connaissances, qui soient autant que possible indépendants d'un langage de représentation/programmation. Le succès pratique de ces entreprises est mitigé dans les deux cas : le fossé entre spécification abstraite (de programmes ou de connaissances) et implémentation effective est énorme, et le rêve de la programmation automatique semble encore loin d'être réalisé.

L'approche ascendante est récente dans le milieu du génie logiciel. Elle est représentée par les travaux sur les *frameworks* et les *patterns* (Gamma et al., 1994). Les *frameworks* sont des squelettes d'application qui imposent une conception figée pour une classe d'applications donnée, tout en donnant la possibilité de redéfinir certaines parties du squelette initial. Le plus souvent ces *frameworks* imposent en fait une *boucle de contrôle* générale dont les parties difficiles sont déjà codées, sous la forme le plus souvent de classes abstraites, et qui appelle des « bouts de code » dont l'écriture est laissée aux soins de l'utilisateur, le plus souvent sous la forme de sous-classes concrètes. Les *patterns* sont une capitalisation d'expériences ayant comme but de codifier des solutions considérées comme « bonnes » à certains problèmes récurrents de conception. Les *frameworks* peuvent alors être conçus comme agrégeant des *patterns*, formant ainsi, avec le niveau de base de la programmation une structure à trois niveaux (Cf. Figure 1).

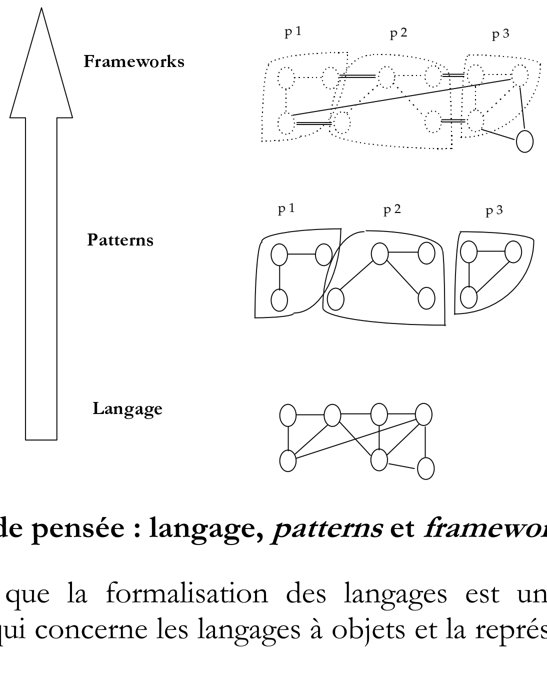


Figure 1. Trois niveaux de pensée : langage, *patterns* et *frameworks*.

Notre point de vue est que la formalisation des langages est un travail souhaitable voire nécessaire, mais qu'en ce qui concerne les langages à objets et la représentation de connaissances, il est encore trop tôt : la plupart des problèmes de représentation difficiles comme celui de la représentation du temps, des hiérarchies de parties, des contraintes, etc. sont encore hors de portée de la totalité des langages objets, qu'ils soient de PPO, de RPO ou des logiques de description. La formalisation de tout ou partie de ces langages est prématurée, beaucoup de travail reste à faire pour trouver de nouveaux cadres dans lesquels les connaissances seront plus aisément exprimées. Pire encore, une formalisation *a priori* nous semble dangereuse : en postulant la suprématie de tel ou tel aspect de la représentation on risque de passer à côté des vrais problèmes, comme le souligne Steward Russel :

> « There is a very heavy responsibility that we do not define away the problem of intelligence. That would be a very terrible thing to do, because then the field would become rather like other fields, where a formal problem has replaced an informal problem, and the interesting parts have been defined away so that solutions become uninteresting, usually highly mathematical and completely ignored by people who have real problems to solve »
>
> Steward Russel, "Rationality and Intelligence", allocution pour la remise du computers and thoughts award à IJCAI 95.

Le cadre de travail que nous adoptons est celui des *frameworks*, qui nous semble adéquat pour supporter notre démarche expérimentale et ascendante, et qui permet de bénéficier de tout un existant technologique sans faire de postulat sur la nature de la connaissance.

2.4. Une boucle générale de contrôle

Le compromis entre généralité et dépendance vis-à-vis d'un domaine se retrouve dans le problème général de la boucle de contrôle. L'histoire de l'IA est peuplée de systèmes proposant des boucles de contrôles fascinantes, comme celle du programme *AM* (boucle de contrôle sans entrée !) (Lenat & Brown, 1984), celle *d'Alice* qui permet de combiner force brute et intelligence (Laurière, 1976), ou encore celle de Soar qui propose un mécanisme général de régression universelle (Newell, 1990). Barbara Hayes-Roth définit même un programme d'intelligence artificielle comme un programme dans lequel se pose le problème du choix de l'opération à effectuer, à chaque cycle de base (Hayes-Roth, 1985). Cette quête d'une boucle de contrôle parfaite et générale est dirigée par le même souci de proposer un mécanisme de raisonnement qui soit à la fois universellement applicable et qui permette l'expression de caractéristiques locales, dépendantes du domaine.

Le programme *CopyCat* (Hofstadter, 1995), est représentatif de cette lignée de grands programmes d'IA. Le but de CopyCat est de trouver des interprétations de séquences de nombres, pour en trouver des prolongements logiques. Par exemple, l'analyse de la séquence (2 4 6 8) par CopyCat produit le chiffre 10, etc. Des séquences plus complexes sont ainsi décryptées par le programme, qui intègre des considérations analytiques (analyser le début de séquence fourni), et prédictives (trouver une raison pour la séquence, qui satisfasse des critères de minimalité, des critères esthétiques, etc.). La description de la boucle de contrôle de CopyCat montre bien vers quel idéal tend cette recherche de la boucle de contrôle parfaite (p. 34) :

> "... Depth-first and breadth-first search, specially when augmented by well-designed and prudent pruning techniques, are both good ideas; they represent two important polar-opposite strategies for searching in huge spaces. However, as I was planning the design of my program, it seemed to me that both of them were far too rigid, and very unlike what people do. I felt that what people do - or at least what I do - is more like an initial very shallow breadth-first scan followed by a bit of depth-first in a local area highlighted by the breadth-first scan, then resurfacing for more of a broad overview, then plunging back in more deeply somewhere, re-emerging for a brief overview

> again, plunging back in perhaps somewhere else for a bit, and so on. In short, a constant interplay between episodes that tend toward the deep side and episodes that tend toward the broad side, with a constant willingness to jump out of any given mode and to try something different for a while. There is also no intense resistance to looking again at an area already looked at once or even more times, because one comes back a second or third time with fresh new eyes..."

Notre position est intermédiaire entre ces boucles de contrôle générales inspirées des programmes d'intelligence artificielle, et les boucles de contrôle spécifiques telles qu'on les trouve dans les *frameworks* spécialisés. Si les *frameworks* développés dans le milieu du génie logiciel à objets permettent de représenter des boucles de contrôle bien conçues, ils ne permettent en général pas de représenter des connaissances du domaine autrement que de manière procédurale, par sous-classes concrètes et ajout ou redéfinition de méthodes. Inversement, les programmes d'IA offrent rarement la possibilité de redéfinir localement des parties du programme, et de s'adapter aux domaines. Une partie de notre travail consiste à trouver des cadres permettant de concilier les avantages des deux approches.

2.5. La connaissance aux limites de la programmation

Les deux familles de langages à objets (RPO et PPO) reflètent deux positions extrêmes vis-à-vis de la fameuse intégration entre génie logiciel et intelligence artificielle, en particulier dans le domaine de la résolution de problèmes. Dans un cas (RPO) on envisage représentation de connaissances et déclarativité comme un substitut, vu comme nécessaire, à l'informatique classique, procédurale. De l'autre côté (PPO) la connaissance n'est intelligible qu'a travers les cadres rigides de la programmation, et l'algorithme, même associé aux classes, est roi.

Notre vision est différente. Outre quelques domaines dans lesquels la connaissance à représenter est soit triviale soit très naïve, le savoir-faire et les connaissances accumulées en programmation et en génie logiciel ne sont pas à la portée des techniques d'intelligence artificielle : aucun système à base de connaissances entièrement déclaratif n'est capable de réaliser les tâches accomplies par les algorithmes dans de nombreux domaines clés de l'intelligence artificielle, comme la résolution de problèmes. De notre point de vue, le rôle de la représentation de connaissances est de s'intercaler dans les trous laissés par ces technologies, qui fournissent l'essentiel de la résolution du problème, en épousant la structure imposée par ces techniques. En d'autres termes, là ou les algorithmes faiblissent, l'ingénieur veut mettre un peu de connaissances. Un de nos buts est alors d'identifier ces trous, et de fournir des cadres et des outils pour les remplir.

2.6. Synthèse

Un des rôles de l'ingénieur "logiciel" moderne consiste à donner la capacité à des systèmes informatiques complexes d'exploiter des connaissances pour les aider à résoudre leurs tâches (mieux ou plus vite). Cette insertion de connaissances doit s'effectuer dans le cadre imposé par le génie logiciel, et concrètement, aujourd'hui, dans le contexte des techniques à objets. Ceci définit une problématique générale de recherche sur la spécification et construction de systèmes à bases de connaissances, dans un contexte défini par :

- Des exigences de **réutilisation**, à la fois de code (bibliothèques de classes existantes), mais aussi de schéma de conception et de boucles de contrôle (les *frameworks* et les *patterns*).

- Une contrainte **d'adaptabilité** au domaine : on veut adapter facilement des programmes généraux à des domaines particuliers, sur lesquels on a des connaissances à représenter.

- Un contexte **technologique** et algorithmique : on veut exploiter l'existant en matière d'informatique « traditionnelle », en particulier en recherche opérationnelle et en résolution de problèmes.

- Une démarche **expérimentale** ascendante, basée sur la construction de systèmes (dans mon cas, des applications en médecine et en musique) puis leur généralisation, plutôt que sur une démarche *a priori*.

Mes travaux se déclinent selon ces quatre axes. J'ai étudié l'intégration de mécanismes d'intelligence artificielle au sein de langages de programmation par objets (règles de production, avec le système *NéOpus* et contraintes avec le système *BackTalk*), à la fois du point de vue technique (implantation efficace) et sous l'angle de la réutilisation. En NéOpus, une des contraintes d'implémentation était de pouvoir réutiliser des classes quelconques (y compris les classes dites système), et de pouvoir exprimer des messages quelconques dans les prémisses et les parties action des règles. J'en ai étudié les conséquences théoriques (le problème de la notification en particulier) et pratiques par le développement de nombreuses applications. En BackTalk, le même problème de réutilisation se pose, et nous a conduit à une démarche particulière d'intégration des contraintes avec les objets (le modèle à classe, que l'on oppose au modèle à attribut). De plus, BackTalk est conçu pour pouvoir réutiliser des algorithmes de satisfaction de manière interchangeable. Enfin, le système *EpiTalk*, réalisé avec la Télé-Université obéit aussi à des contraintes fortes de réutilisation de systèmes existants, sur lesquels on greffe une couche de représentation des tâches usuelles, en vue de la production rationnelle de conseils pertinents.

La réalisation de plusieurs applications utilisant ces différentes techniques m'a permis de dégager des similarités dans différents corpus de connaissances (par exemple les *patterns* de règles obtenus par comparaison de systèmes de raisonnement temporel en musique et en médecine). De nombreuses applications musicales bâties autour de la bibliothèque de classes *MusES* m'ont par ailleurs donné l'occasion d'explorer d'autres techniques d'intelligence artificielle (classification, systèmes de raisonnement à partir de cas) et leur intégration dans un univers logiciel relativement complexe.

3. Objets et règles

Mon travail de thèse concernait l'étude de l'intégration de mécanismes de règles de production en chaînage avant dans un langage de programmation par objets, préoccupation directement issue de problèmes de représentation rencontrés lors du développement du système d'improvisation musicale. Je décris ici quelques travaux qui s'inscrivent dans la continuation directe de cette thèse, après en avoir rappelé brièvement les résultats principaux.

3.1. NéOpus

Le point de départ de mon travail fut la réécriture du système OPUS, à partir de la description du papier de Atkinson et Laursen (Atkinson & Laursen, 1987). Ce système était le seul à proposer une intégration d'un langage objets avec un mécanisme de règles à la OPS-5, c'est-à-dire avec une compilation efficace sous forme de réseau RETE (Forgy, 1982). L'ensemble de ce travail est décrit dans ma thèse (Pachet, 1992a), et son fonctionnement pratique dans (Pachet, 1991c; Pachet, 1991e; Pachet, 1995a; Pachet, 1995b; Pachet & Perrot, 1994a). Le système est utilisé dans un nombre appréciable d'applications (voir 3.7). Il a été l'objet d'une nouvelle implémentation par la société OTI (Object Technology International, Ottawa), sous le nom d'ENVY/Expert (Barry & McAffer, 1992), et est utilisé en interne pour le développement d'applications. Je résume ici les points les plus saillants de ce système.

3.2. Le problème des métaclasses

1. *Petite histoire des métaclasses Smalltalk*

Le modèle ObjVlisp de Pierre Cointe (Cointe, 1987) a fortement influencé la recherche française en matière de langages à objets. Un des points les plus frappants de ce modèle est le fait qu'il permet de définir des métaclasses en toute généralité, sans contrainte, et ce, dans un modèle uniforme, élégant et complet. En revanche, le langage Smalltalk propose une architecture à métaclasse hybride, dans laquelle l'utilisateur n'a que des possibilités limitées de définir des métaclasses, mais assure en contrepartie un *héritage parallèle* des métaclasses aux classes. Cet héritage parallèle permet de garantir l'héritage des méthodes de création aux sous-classes. Une extension de Smalltalk pour manipuler des métaclasses à la ObjVlisp a été réalisée par Jean-Pierre Briot et Pierre Cointe, avec le système *ClassTalk* (Briot & Cointe, 1989), transposition de l'architecture à métaclasses *ObjVlisp* en Smalltalk.

Néanmoins, une critique des architectures à métaclasses "complètes", à la ObjVlisp, fut formulée par Nicolas Graubé (Graubé, 1989), qui a exhibé des problèmes de compatibilité pouvant survenir lorsque l'utilisateur choisit ses métaclasses indistinctement. Récemment, une réponse à ces critiques à été proposée par (Danforth & Forman, 1994), basée sur la création dynamique de métaclasses assurant la compatibilité au sens de Graubé. Lors du *workshop* que j'ai co-organisé avec Hafedh Mili à OOPSLA 95 (Mili et al., 1995), il est apparu que la solution de Forman impose certaines restriction sur le modèle. On ne peut pas, par exemple, produire une métaclasse représentant les classes abstraites, c'est-à-dire n'ayant pas de méthode de création `new` (critique de Pierre Cointe). La réponse de Forman fut alors que l'absence d'une méthode `new` est une propriété non héritable, car un modèle basé sur l'héritage ne peut factoriser que des propriétés "positives".

2. *Métaclasses et NéOpus*

L'architecture proposée par Atkinson & Laursen repose sur une réification très astucieuse des *tokens*, c'est à dire des n-uplets d'objets devant vérifier les conditions des règles dans l'algorithme RETE. A chaque base de règles est associée une classe "fantôme", de manière bijective. Cette classe représente les tokens devant vérifier les conditions des règles de la base. L'idée maîtresse est ensuite de compiler les règles de telle manière à ce qu'à chaque condition de règle corresponde une méthode pour cette classe fantôme. Vérifier qu'un token satisfait une condition de règle se traduit alors par un simple envoi de message au token. On gagne alors en simplicité de conception et en efficacité, la méthode étant compilée par le compilateur Smalltalk, évitant ainsi de coûteux passages de paramètres.

L'implémentation de ce principe, telle que proposée par les auteurs posait cependant un problème. Celle-ci, en effet, reposait sur la construction d'une métaclasse particulière `OpusTokenBehavior`, dont les instances étaient les classes de tokens (cf. Figure 2, gauche). Or l'architecture de métaclasses standard de Smalltalk ne permettait pas (et ne permet toujours pas) à l'utilisateur de programmer explicitement ses propres métaclasses.

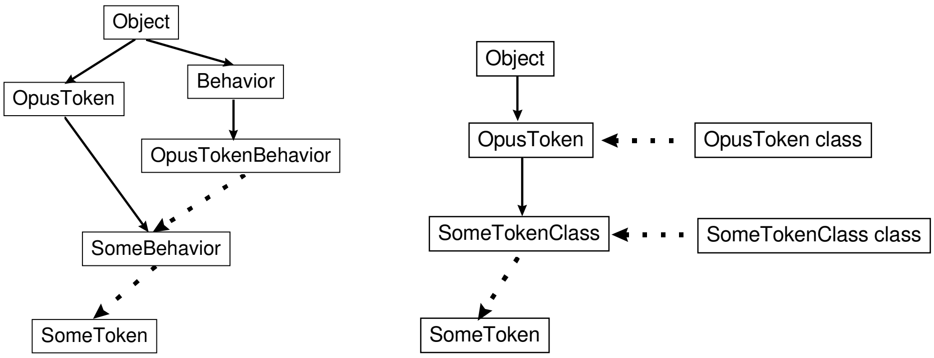


Figure 2. Le schéma d'implémentation des classes de tokens en Opus. A gauche, tel que décrit par les auteurs du système original. Les flèches en plein désignent la relation d'héritage ; en pointillé celles d'instanciation. A droite, l'architecture effectivement implémentée dans NéOpus avec les métaclasses Smalltalk standard.

Cette remarque fut l'occasion de rencontrer Jean-Pierre Briot et Pierre Cointe, au moment où ils travaillaient sur l'architecture *ClassTalk* J'ai donc tenté d'utiliser ClassTalk pour résoudre le problème des classes de Token. Malheureusement, les métaclasses ClassTalk formaient un univers à part, non intégré à l'univers standard Smalltalk. En pratique, il n'était pas possible de fabriquer des sous-classes "normales" de classes ClassTalk, rendant impossible l'utilisation de ce mécanisme sans une refonte complète du système[3].

J'ai donc du abandonner le schéma initial proposé par les auteurs (ceux-ci avaient bien du résoudre ce problème, mais leur solution est encore aujourd'hui une énigme pour moi!), et en proposer un autre, plus facile à implémenter dans l'architecture standard (cf. Figure 2 droite).

[3] Les travaux récents de Frédéric Rivard (Ecole des Mines de Nantes) devraient proposer une nouvelle version de ClassTalk (NéoClassTalk) permettant enfin l'intégration des deux mondes.

Ces travaux m'ont fait comprendre qu'une architecture théoriquement meilleure, comme celle de ClassTalk, n'était pas forcément souhaitable en pratique. En particulier, l'héritage parallèle des métaclasses, tel qu'implémenté en Smalltalk-80, s'est révélé être, malgré ses limitations théoriques, un excellent compromis entre flexibilité et expressivité. J'ai proposé quelques utilisations des métaclasses Smalltalk standard dans (Pachet, 1989), notamment pour le problème de la sauvegarde d'objets, mais cette question des métaclasses n'est pas pour moi terminée (voir 8.1.1). Bref, le débat n'est pas CLOS!

3.3. Le problème de la notification

Un point important de l'intégration objets et règles telle que proposée dans le système OPUS, et souligné par les auteurs, est le problème de la notification, qui peut être résumé comme suit. On peut trouver dans la partie conclusion d'une règle des envois de messages (transmissions) quelconques. L'exécution d'une de ces transmissions va probablement modifier un certain nombre d'objets. Ces modifications vont à leur tour modifier la valeur de vérité des conditions des règles de la base, et ainsi rendre certaines règles déclenchables (ou au contraire rendre certaines règles qui étaient déclenchables non déclenchables). Le problème est d'identifier automatiquement ces modifications de manière à maintenir la cohérence de l'ensemble des conflits (*conflict set*).

Lorsque les parties *conditions* et *action* des règles sont de simples contraintes sur des valeurs d'attributs, ce calcul ne pose pas de problème car il suffit de relier les parties actions avec les parties condition qui portent sur des attributs en commun. En Opus, ce lien est caché, par définition, en vertu du principe d'encapsulation. La solution adoptée par Atkinson & Laursen, et reprise dans NéOpus, consiste à demander explicitement à l'utilisateur de déclarer les objets modifiés par l'application d'une règle. Cette déclaration se traduit par l'envoi d'un message (`modified`) à l'objet modifié. Ce message est au cœur du moteur; c'est lui qui permet de mettre à jour les états d'instanciation des règles et d'assurer le bon fonctionnement du cycle de base.

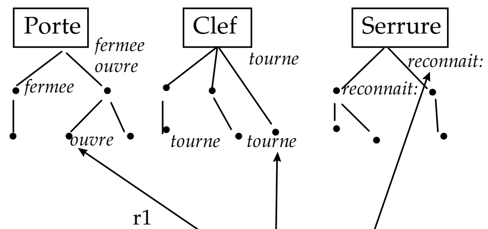
J'ai organisé un workshop sur le thème des EOOPS (Embedded Object Oriented Production Systems[4]) à OOPSLA en 1994 (Pachet, 1994c; Pachet, 1994d). Ce workshop rassemblait les chercheurs principaux dans ce domaine : C. Forgy, D. Miranker, P. Albert, B. Barry, J. Bouaud et R. Crawford. Si chaque chercheur propose une intégration particulière des règles avec un langage à objets, le problème de la notification est systématiquement rencontré et aucune solution n'a été trouvée pour l'instant. Partant d'une critique de la solution NéOpus, les travaux plus récents de Bouaud et Voyer proposent une solution théorique (Bouaud & Voyer, 1996), mais l'implémentation est encore incomplète.

3.4. Trois mécanismes : typage naturel, contrôle et héritage de bases de règles

Les originalités principales du système NéOpus sont l'introduction de trois mécanismes : le typage naturel, l'héritage de bases de règles, et le contrôle par méta-règles. Ces trois mécanismes sont en principe indépendants, mais il se sont trouvés en pratique être utilisés conjointement, de manière systématique.

[4] Ce workshop est à l'origine de l'acronyme EOOPS...

D’une part, le typage naturel consiste à proposer un typage des variables de règles qui prennent en compte l’héritage de classe. Nous avons montré que ceci permet de réduire le nombre de règles à écrire dans le cas de règles portant sur plusieurs variables. D’autre part, ce mécanisme permet de doter les règles d’une capacité d’abstraction, leur sémantique étant donnée non pas par leur texte seul mais par l’ensemble {texte, contexte des classes dont les objets sont instances}. Dans l’exemple de la Figure 3, une seule règle définit le comportement d’un triplet d’objets (une porte, une clef et une serrure). Chacun de ces objets peut instancier différentes sous-classes de leur classe de définition, et cette sous-classe peut redéfinir certains messages utilisés dans les prémisses ou la conclusion de la règle. La règle représente donc potentiellement un produit cartésien de règles.




```
r1
| Porte p. Clef c. Serrure s |
Si
   p fermee.
   s reconnaît: c.
Alors
   c tourne. p ouvre
```

Figure 3. Le typage naturel en NéOpus. Chaque variable peut être instanciée par une sous classe concrète de la classe de déclaration. Pour 3 variables et 5 classes concrètes possibles par variable, le nombre total de règles concrètes est de 5^3=125.

D'autre part, le mécanisme d’héritage normal des classes a été réinterprété pour structurer les bases de règles. J'ai montré que ce mécanisme permettait de factoriser des règles communes à plusieurs base de règles, tout en lui donnant une sémantique opérationnelle raisonnable. Partant du parallèle méthode/règle puis classe/base de règles, le problème principal ici était de donner une interprétation au mécanisme de *lookup* pour les bases de règles. En effet, celui-ci n'a de sens que parce que les méthodes sont appelées explicitement. Dans un univers de règles en chaînage avant, les règles ne sont plus exécutées par invocation du nom, mais par filtrage. Il est alors impossible de transposer l'intuition de l'héritage telle que fournie par le mécanisme du lookup. J'ai proposé d'associer à l'héritage des bases de règles une stratégie de contrôle qui transpose l'intuition de l'héritage dans le monde des règles, en préférant systématiquement les règles définies dans les bases les plus basses de la hiérarchie (Pachet, 1992b).

Le troisième mécanisme propre à NéOpus est l'architecture de contrôle. Celle-ci permet de spécifier le contrôle d'une base de règles à l'aide d'une autre base de règles. L'activation d'une base de règles est alors entièrement réflexif (Pachet, 1991a), (Pachet & Perrot, 1994b), (Pachet, 1991d). Ce mécanisme permet de produire des bases de méta-règles qui définissent les stratégies standard d'activation. Ces bases de règles manipulent des objets particuliers, appelés *évaluateurs*, qui ne sont rien d’autre que des réifications de l’état d’activation d’une base de règles. Le mécanisme d'héritage permet de structurer ces bases de méta-règles et de faciliter ainsi leur écriture. Une des applications les plus frappantes de cette architecture est le système NéoGanesh de Michel Dojat (voir 3.7).

3.5. Distribution et concurrence

Une fois les mécanismes de base mis en place, l'idée de rendre les bases de règles NéOpus concurrente est naturelle, voire obligatoire : difficile de résister à l'air du temps! Les premières expérimentations dans ce sens furent motivées par Les Gasser, qui m'a proposé une base de règles traitant de la propagation de substances toxiques. L'implémentation de cette expertise sous forme de bases de règles NéOpus concurrentes permit de valider cette idée sur un exemple simple, et de poser un certain nombre de problèmes, dont celui du partage d'objets entre bases de règles. La thèse de Zahia Guessoum aborde ce problème de manière systématique, et propose une solution fondée sur l'exploitation d'un graphe de dépendances (Guessoum, 1996). De plus, chaque base de règles est encapsulée dans un agent, qui possède son propre module de contrôle, représenté par un ATN (Augmented Transition Network). L'architecture de ce modèle multi-agents, appelé DIMA, permet à la fois l'asynchronisme au sein d'un agent (qui peut en même temps percevoir des informations, raisonner et agir), et entre agents. Enfin, l'implémentation de DIMA fournit une application intéressante du mécanisme d'héritage de bases de règles, pour simuler une sorte de mécanisme *anytime* (Zilberstein & Russel, 1996) : les différents raffinements d'un même calcul sont représentés par une hiérarchie de bases de règles, chaque sous-base redéfinissant tout ou partie du calcul.

Jeff McAffer aborde le problème de la concurrence des bases de règles NéOpus en prenant un point de vue plus général : celui de la méta programmation (Briot & Guerraoui, 1996). Le système *CODA* développé par McAffer est une architecture réflexive pour le prototypage, permettant la redéfinition de la sémantique computationnelle de programmes objets (McAffer, 1995). Une des applications de CODA est l'extension de bases de règles NéOpus, dans leur format ENVY/Expert, pour les rendre concurrentes et distribuées. McAffer montre que cette extension, grâce à son système, peut se faire en un minimum de modifications du code source original.

3.6. S'échapper des règles

Malgré les avantages indéniables qu'apporte la programmation par règles, surtout quand elle est couplée avec la programmation par objets, la conception et le développement d'applications en NéOpus est difficile. Le système est général, et comporte beaucoup de degrés de liberté. La démarche ascendante que nous suivons nous a fait naturellement rechercher des régularités dans les bases de connaissances développées en NéOpus. Ces réflexions ont donné lieu à deux types de résultats.

1. Patterns d'EOOPS

Un premier travail consiste à s'inspirer de l'idée des *patterns* (Gamma et al., 1994). Pour ces auteurs, un *pattern* est une solution à un problème de conception récurrent. Cette solution est exprimée de manière informelle (en fait essentiellement un dessin OMT accompagné de commentaires).

Nous pensons que ce qui fait la substance des *patterns* est la présence de trois mécanismes - instanciation, héritage, et agrégation - qui sont à la fois orthogonaux et complémentaires. Ces trois mécanismes sont suffisamment orthogonaux pour donner lieu à une énorme variété de combinaisons possibles. Par ailleurs, ils sont suffisamment complémentaires pour pouvoir produire des solutions opérationnelles à des problèmes de conception.

Le lien avec les travaux sur les *frameworks* et les *patterns* est ici direct. Il consiste à postuler que les trois mécanismes de base mis en œuvre dans NéOpus (typage naturel, héritage de bases de règles et contrôle par méta-règles) sont du même ordre que les trois mécanismes de base de la PPO. Ces trois mécanismes, dégagés en particulier dans (Dojat & Pachet, 1995a), sont à la fois orthogonaux et complémentaires. Dès lors nous nous intéressons à identifier des *patterns* d'EOOPS, cristallisant des solutions pour représenter certain schémas d'inférence récurrents, à l'aide de ces trois mécanismes.

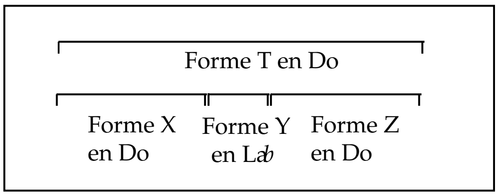
La comparaison de deux applications NéOpus - un système de suivi ventilatoire de patients (4.5) et un système d'analyse harmonique (7.2.1) - a permis de dégager plusieurs *patterns* d'EOOPS (Pachet & Dojat, 1995), (Pachet, 1996). A titre d'exemple le *pattern* « Reconnaissance » est typiquement utilisé pour représenter le mécanisme d'agrégation pour le modèle temporel de Michel Dojat. Ce pattern, qui met en œuvre un seul type de règle, peut se décrire informellement comme suit :

- La règle filtre un certain nombre d'objets de base, tous ayant une super classe commune,
- La règle a comme partie action la création d'un objet, lui-même ayant la même super classe commune que les objets filtrés,
- L'objet créé en partie action agrège les objets filtrés,
- La règle est récursive : l'objet créé peut lui-même participer à l'instanciation de la même règle.

Le *pattern* peut se décrire comme suit, dans une pseudo-syntaxe simple :

Soient `a1 a2` instances de (une sous-classe de) `ClasseRacine` SI certaines propriétés sur l'agencement de `a1` et `a2`, et certaines propriétés sur leurs caractéristiques locales sont vraies ALORS **Créer** une instance d'une sous-classe de `ClasseRacine` couvrant la durée des deux objets agrégés. **Etablir** le lien de composition entre le nouvel objet et les deux anciens.

Ce *pattern* s'instancie pour les deux systèmes (analyse harmonique et le suivi respiratoire). La règle d'emprunt modal (Figure 4, gauche) indique qu'une modulation passagère peut être considérée comme non significative, lorsqu'elle est au milieu d'un passage continu, et qu'elle vérifie un certain nombre de propriétés harmoniques (elle peut être analysée dans la tonalité mineure de la tonalité des deux formes qui l'entourent). De manière similaire, la règle de continuité par partie (Figure 4, droite) indique qu'une instabilité ventilatoire de courte durée peut être considérée comme non significative lorsqu'elle est au milieu d'une ventilation stable.



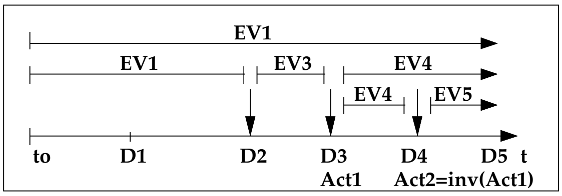


Figure 4. Deux règles de reconnaissance, présentées ici de manière graphique. A gauche une règle d'emprunt modal. Une forme globale en Do persiste sur l'ensemble des intervalles de temps. A droite, une règle de continuité par partie. L'état ventilatoire EV1 persiste malgré l'apparition de différents états intermédiaires qui entraînent les actions Act1 et Act2 sur le respirateur.

D'autre *patterns* ont été dégagés, comme celui de « destruction », correspondant aux règles d'oubli, et d'autres plus complexes faisant intervenir plusieurs règles, et le mécanisme d'héritage de bases de règles. L'objectif principal de ce travail, en cours, est idéalement l'identification d'un petit nombre de *patterns* d'EOOPS qui permettent de couvrir une grande partie des besoins des concepteurs d'applications par objets et règles. Une fois identifiés, ces *patterns* peuvent alors donner lieu à des outils spécialisés pour rentrer les connaissances sous-jacentes, et aboutir ainsi à des systèmes moins généraux que NéOpus, mais adaptés à des besoins particuliers de l'utilisateur.

2. *Le conçu et le perçu*

Une autre idée développée dans (Pachet, 1994f) a consisté à mettre en avant la distinction, dans les bases de règles à objets, entre deux types d'objets au statuts ontologiques différents. D'une part, les objets *perçus* modélisent le monde sur lequel travaille l'expert. D'autre part les objets manipulés lors du raisonnement constituent le monde *conçu*. Cette distinction perçu/conçu ne recouvre pas nécessairement la distinction concret/abstrait, étant purement locale au problème traité. Ainsi, un objet pourra-t-il être perçu par un expert pour un tâche donnée, et conçu pour une autre. Un des intérêts de cette distinction est qu'elle conduit à un modèle du processus de raisonnement dans un EOOPS, dans lequel les règles ont un statut bien défini : à partir de configurations du monde conçu, elles doivent animer un monde perçu.

3.7. Applications de NéOpus

L'application princeps de NéOpus est le système NéoGanesh développé par Michel Dojat pour sa thèse (Dojat, 1994). Ce système, qui fonctionne en boucle fermée, a pour but de contrôler des respirateurs dans les unités de soin intensifs. Ces appareils produisent une aide respiratoire à des patients incapables de respirer tout seuls (voir Figure 5). Leur réglage nécessite une expertise médicale complexe, faisant intervenir un raisonnement temporel abstrait à partir de données brutes recueillies sur le patient (pression de co2 de fin d'expiration, volume inspiré et fréquence respiratoire). De plus, le système effectue un raisonnement spécial dans le but de « sevrer » les patients lorsque c'est possible, de manière à limiter l'accoutumance. Des validations cliniques ont montré que le système obtenait des performances très satisfaisantes (Dojat et al., 1996). Ce système est aujourd'hui un des rares systèmes d'intelligence artificielle médical opérationnel et utilisé en routine dans un hôpital. Sa validation multicentrique a fait l'objet d'une proposition de projet Européen (ALPS, programme Bio-med-II). Le but ultime d'une telle validation est d'intégrer NéoGanesh directement sur la puce de contrôle du respirateur.

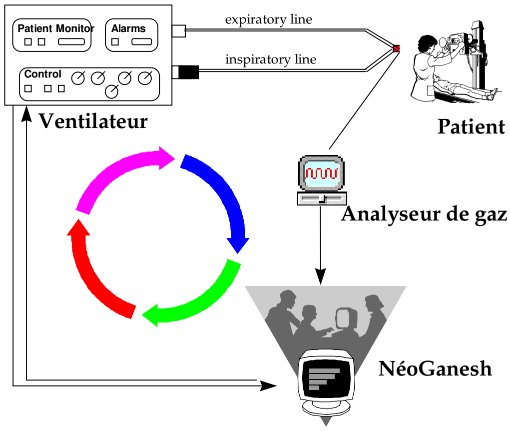
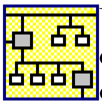


Figure 5. Le système NéoGanesh en action.

Une synthèse complète de ce projet peut être trouvée dans (Dojat et al., 1997). Du point de vue de la représentation des connaissances, le système NéoGanesh met en œuvre tous les mécanismes de NéOpus : héritage de bases de règles, contrôle déclaratif, typage naturel des variables. En outre, un modèle temporel a été introduit par Michel Dojat, fondé sur deux types particulier de règles : agrégation et oubli (Dojat & Sayettat, 1996). Ce modèle temporel s'implémente aisément en NéOpus grâce à la combinaison des trois mécanismes de base. L'application a été l'objet de plusieurs développements et réflexions. Les aspects appliqués, et l'intégration dans un milieu réel ont été décrits dans (Dojat & Pachet, 1992c). L'aspect réutilisable de l'architecture a été mis en avant dans (Dojat & Pachet, 1992a; Dojat & Pachet, 1992b). En particulier, nous avons montré comment la combinaison des mécanismes de représentation de NéOpus permettait d'atteindre une « réutilisabilité limitée », que nous avons comparée avec les positions épistémologiques courantes en informatique médicale (Dojat & Pachet, 1995a). Nous avons par ailleurs développé l'idée que cette architecture permettait une représentation implicite, mais opérationnelle du contexte (Dojat & Pachet, 1995b; Dojat & Pachet, 1995c).

Plus récemment, une version véritablement multi-agents de NéoGanesh a été réalisée par Zahia Guessoum comme application de son *framework* DIMA. Cette réécriture a comme but de rendre les différents modules de NéoGanesh concurrents, afin de rendre le système plus réactif aux alarmes.


Une deuxième application importante de NéOpus est le système *MétaGen*, développé dans l'équipe Métafor du Laforia (Blain et al., 1994; Revault et al., 1994). MétaGen est un outil incarnant les idées de Gilles Blain sur la métamodélisation. Partant de l'échec relatif des ontologies réutilisables pour les objets dits « métiers » (i.e. non techniques), la métamodélisation consiste à prendre le contre-pied de la vision « composants réutilisables » : ce qu'on cherche à faire est non pas de modéliser des objets, des classes ou des hiérarchies de concepts, mais de modéliser les langages dans lesquels ces concepts peuvent être décrits. Il s'agit alors de mettre en place un dialogue d'un type particulier dans lequel l'expert et l'informaticien s'expriment dans deux langages différents, et où le système a comme tâche de transformer l'un en l'autre. Les modèles en Métagen sont exprimés par des classes Smalltalk (générées automatiquement à partir de descriptions graphiques), et NéOpus est utilisé pour spécifier les transformations entre ces modèles. Le séquencement des transformations est naturellement spécifié par des métabases. L'implémentation complète de MétaGen fait l'objet de la thèse de N. Revault (Revault, 1996). MétaGen est utilisé pour modéliser divers systèmes d'informations, en particulier dans le domaine des bases de données (thèse de H. Sahraoui, (Sahraoui, 1995)).


NéOpus est utilisé dans plusieurs applications musicales développées autour du système MusES (décrit en détail en 7.3). Le système d'analyse harmonique (Mouton & Pachet, 1995; Pachet, 1994f) (voir 7.2.1), est une application importante de NéOpus dans le domaine du raisonnement temporel, que nous comparons avec la partie temporelle du système NéoGanesh (Cf. 7.2.1). Le système de simulation d'improvisations de Geber Ramalho (Ramalho, 1996), basé sur une modélisation de la mémoire musicale sous forme de cas, utilise plusieurs bases de règles NéOpus pour identifier et combiner des cas. Toujours dans le domaine du raisonnement à partir de cas, signalons aussi la thèse de Sophie Rougegrez (Rougegrez, 1994) sur l'utilisation de ces techniques pour la prédiction d'incendies, et qui utilise NéOpus pour transformer des cas.


Deux thèses ont été menées pour étudier l'extension de NéOpus vers la distribution et la concurrence. Celle de Z. Guessoum (Guessoum, 1996) propose de réutiliser le modèle Actalk de Jean-Pierre Briot (Briot, 1989a) pour modéliser des « agents NéOpus ». Les problèmes de synchronisation de bases de règles y sont traités par la gestion d'un réseau de dépendances entre bases de règles. La thèse de (McAffer, 1995) suit une autre direction et propose une architecture réflexive permettant de modifier la sémantique opérationnelle d'applications existantes. Son système est illustré notamment par une version concurrente de NéOpus (voir 3.5).


NéOpus est utilisé dans divers projets de systèmes tutoriels. Le système *Diapason* a été développé pour l'apprentissage du diagnostic de pannes dans les réseaux EDF. NéOpus y est utilisé pour le module de résolution de problèmes (Joab et al., 1995; Moinard, 1994). Le système *CardExp* est un système expert traditionnel développé à l'Université de Montréal, pour la détection et le suivi de maladies cardio-vasculaires (Dufresne et al., 1995). La base de règles NéOpus comporte environ 300 règles. Un projet en cours consiste à connecter cette base de règles avec des bases de données de cas réels de l'hôpital de Montréal. Le projet *Safari* de l'Université de Montréal utilise NéOpus pour divers modules dont celui de génération de curriculum (NKambou, 1995) (NKambou et al., 1995). Enfin NéOpus est utilisé dans l'architecture EpiTalk pour engendrer des conseils (voir section 4).

Sur une échelle plus réduite, NéOpus a été utilisé pour divers projets de recherche (thèses et DEA). Philippe Laublet a utilisé NéOpus pour construire un langage de représentation de connaissances mathématiques dans son système *ForrEnMat* (Laublet, 1993; Laublet, 1994). NéOpus a été utilisé pour piloter un système d'analyse d'images tomoscintigraphiques à l'hôpital de Nantes (Forte, 1991; Forte et al., 1992). Isabelle Alvarez a développé un modèle d'« explications différentielles » en expérimentant avec des bases de règles NéOpus dans le domaine de l'agriculture au CEMAGREF (Alvarez, 1992). La thèse de (Benyahia, 1994) utilise NéOpus pour modéliser des systèmes d'exploitation temps réel. Amel Benhouhou modélise des connaissances dans un système tutoriel intelligent pour l'aide à la décision dans les situations de crise avec NéOpus (Benhouhou, 1993; Benhouhou, 1996). Odile Fillod a montré que l'on pouvait construire des systèmes s'observant pour générer des règles dynamiquement afin de s'améliorer (Fillod, 1995). Plusieurs étudiants ont participé au développement d'aspects particuliers du système comme l'implémentation des nœuds négatifs (Charbonnel, 1990). Enfin, une application de NéOpus à la géométrie a été décrite dans (Pachet, 1990b).

4. Systèmes conseillers et reconnaissance de plans

J'ai eu l'occasion de travailler avec Sylvain Giroux et Gilbert Paquette au LICEF[5], lors de mon stage post doctoral en 1992, et régulièrement depuis. Le LICEF est un cadre idéal pour expérimenter de nouvelles formes d'éducation dans lesquelles l'ordinateur joue un rôle d'assistant de plus en plus intelligent, capable d'analyser rapidement des situations complexes. Le problème de départ était la construction de systèmes "conseillers" dans les domaines des systèmes tutoriels collaboratifs et distribués. Le laboratoire LICEF avait déjà accumulé une expérience considérable dans la fabrication de tels systèmes conseillers, en particulier par les travaux de Gilbert Paquette (Paquette, 1991). Une des contraintes majeures était la réutilisation de systèmes déjà écrits.

4.1. Genèse du projet EpiTalk, idées principales

La notion de conseil pour un système informatique n'a jamais été formellement définie. Intuitivement un conseil est une action du système qui n'a pas nécessairement de conséquence directe. Dans le cadre des systèmes conseillers à vocation pédagogique, un conseil se matérialise le plus souvent par un texte présenté à l'utilisateur. Le problème est alors de déterminer le contenu de ce texte (le *quoi*), et le moment adéquat et pertinent de le présenter (le *quand*).

Par ailleurs le LICEF avait comme contrainte supplémentaire la réutilisation d'applications existantes, non forcément conçues pour être utilisées dans un contexte pédagogique. Ces contraintes nous ont amené à élaborer progressivement une certaine conception des systèmes conseillers, fondé sur une analogie botanique : les plantes épiphytes. Les plantes épiphytes sont des plantes qui ont besoin d'un système hôte pour vivre, mais qui ne le détériorent pas, au contraire des parasites, ou, pire, des prédateurs. Le lierre, certaines orchidées sont des plantes épiphytes. Par analogie, nous avons introduit la notion de *système épiphyte*, comme un système dont l'activité consiste à observer un autre système sans le perturber, et à analyser ces observations pour produire un certain type d'abstractions, que l'on appelle conseils.

Le système résultant, appelé EpiTalk (dans une longue tradition de systèmes aux noms postfixés par *Talk*) est le résultat de recherches menées en collaboration étroite avec Sylvain Giroux et Gilbert Paquette. Ce - gros - système est décrit d'un point de vue général dans (Giroux et al., 1995; Giroux et al., 1996; Pachet et al., 1994; Paquette et al., 1993; Paquette et al., 1994; Paquette et al., 1996).

D'un point de vue technique, les systèmes épiphytes sont basés sur un échafaudage de plusieurs concepts. Au niveau le plus bas, une couche dite d'*espionnage* permet de récupérer les informations brutes du système observé. Ces informations sont ensuite analysées par rapport à un arbre de tâches. L'analyse produit *in fine* un ou plusieurs conseils, qui sont ensuite eux-mêmes traités avant d'être présentés à l'utilisateur. Je vais ici décrire les points qui me paraissent les plus importants et sur lesquels j'ai particulièrement travaillé : la couche d'espionnage et l'interprétation de l'arbre des tâches.

---

[5] Le LICEF est un des quatre membres du "Consortium for CourseWare Engineering" avec l'Université du Minnesota, l'Université de Bergen (Norvège) et le MIT (Projet Athena-Muse).

4.2. Espions

Des objets particuliers, appelés "espions", se greffent sur les objets clé du système hôte, et interceptent les messages pour les renvoyer au système d'analyse. Ce mécanisme d'espionnage utilise fortement les capacités réflexives du langage Smalltalk (primitives `doesNotUnderstand`: et `0`des conclusions de ce travail, est que la programmation par instance peut être harmonieusement mêlée à la programmation par classe, renouvelant ainsi le vieux débat héritage/délégation (Liebermann, 1986).

4.3. Reconnaissance de plan et interprétation de l'arbre des tâches

La construction de systèmes conseillers repose sur deux postulats :

1. Donner des conseils pertinents implique de connaître les *intentions* de l'utilisateur.
2. Connaître les intentions de l'utilisateur requiert une *analyse* des séquences d'actions temporelles.

Or, la recherche en reconnaissance de plans (point 2), en particulier les travaux inspirés par la logique de circonscription (Kautz & Allen, 1986), montre clairement que le problème de l'analyse n'a pas de solution en général. Une des idées que nous avons explorées pour produire des conseils a été de combiner les deux étapes - analyse des actions et production effective du conseil - en un seul et même mécanisme (Pachet & Giroux, 1995). Ceci permet d'orienter l'analyse vers la production de conseils, son but ultime, d'où une analyse moins fine que ce que peuvent produire certains formalismes, mais dont le résultat est directement exploitable.

Ce mécanisme à double fonction repose sur une représentation arborescente des tâches typiques à reconnaître. Ces *arbres de tâches* sont en principe fournis par des experts pédagogues, habitués à formaliser leur expertise sous une forme codifiée. Les feuilles de ces arbres représentent les actions terminales considérées comme atomiques. Les nœuds intermédiaires représentent les tâches abstraites. L'arbre représente ainsi le lien tâche/sous-tâche (voir exemple Figure 6). Le mécanisme est déclenché à chaque fois qu'un espion détecte des informations dans le système hôte, et consiste à effectuer une remontée des feuilles concernées par les informations espionnées vers la racine de l'arbre. A chaque rencontre de nœud intermédiaire représentant une tâche abstraite, un mécanisme local d'automate fini permet 1) de mettre à jour un modèle de la tâche en cours et 2) de produire éventuellement un conseil.

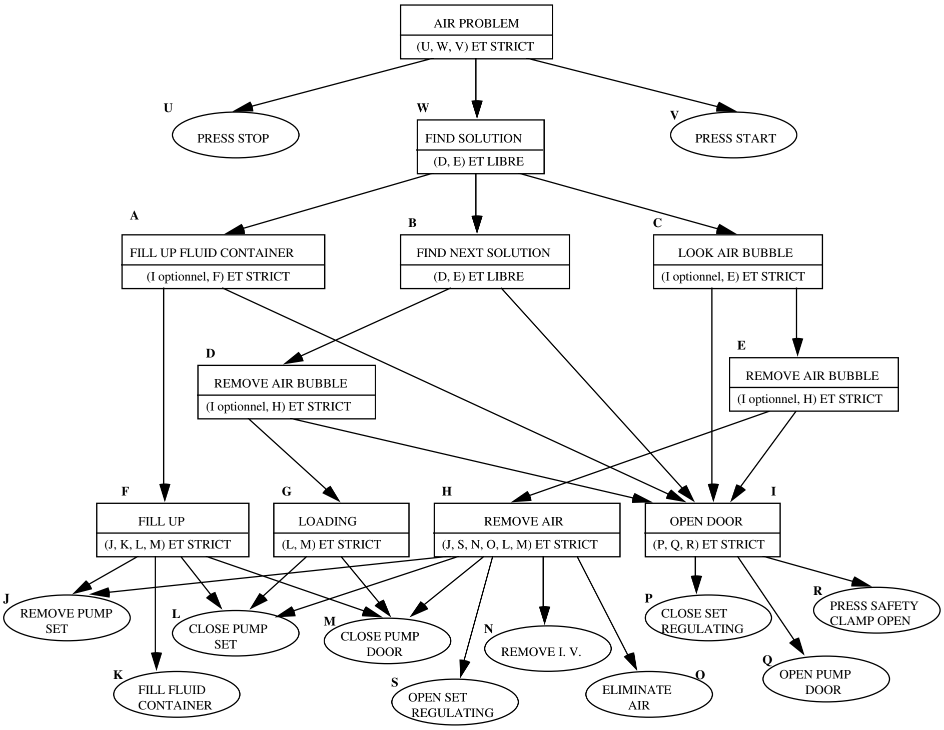


Figure 6 Un exemple d'arbre de tâches utilisé dans le système SAFARI. La tâche modélisée est « détection d'une bulle d'air dans la pompe à perfusion Baxter ©».

D'autres liens ont été ajoutés au modèle de base, comme les liens de précédence temporelle. Dans le cas de l'arbre des tâches de la Figure 6, ces liens permettent d'engendrer des conseils du type « la tâche <Entrer les taux> est effectuée prématurément, vous devriez d'abord appuyer sur le bouton ON/OFF », ou bien, « la tâche <Entrer Volume> a déjà été effectuée, vous pouvez maintenant soit entrer un débit soit entrer un temps », etc.

J'ai montré que l'on pouvait appliquer l'architecture épiphyte de base pour construire un système qui produise de manière systématique tous ces conseils de violation de précédence, indépendamment du domaine, avec une précision plus fine que celle obtenue par les techniques traditionnelles d'appariement (Pachet et al., 1996a), (Djamen & Pachet, 1997). Ce modèle a servi de base au système de modélisation du raisonnement d'étudiants en physique quantitative, développé par J-Y. Djamen dans sa thèse (Djamen, 1995). Le modèle enrichi est en outre utilisé pour une application du système SAFARI (Gecsei & Frasson, 1994) à l'hôpital de Montréal (Alexe & Gecsei, 1996).

4.4. Une architecture multi-agents

Le système EpiTalk a été pensé comme un système multi-agents dès sa conception, et ce sous l'impulsion de Sylvain Giroux, dont la thèse concerne le rapport entre agents et systèmes (Giroux, 1993). Cette thèse propose une définition précise d'un type particulier de réflexivité, dite « organisationnelle », par laquelle les organisations d'agents sont représentées explicitement et donc manipulables, modifiables, et reconfigurables. L'intégration de cette forme de réflexivité avec celle proposée en NéOpus par les méta-règles a été tentée (Giroux et al., 1993), et est d'une certaine manière à l'origine du projet EpiTalk. L'aspect multi-agents d'EpiTalk est en effet central si l'on veut être capable de produire des conseils sur des applications distantes (Giroux et al., 1994b). Une autre application directe de l'aspect distribué de l'architecture EpiTalk est le problème difficile du déverminage (*debugging*) des langages d'acteurs, pour lequel nous avons montré qu'EpiTalk est un bon paradigme (Giroux et al., 1994a).

Il reste cependant des problèmes techniques liés à la distribution du système conseiller. Le principe même de l'interprétation de l'arbre des tâches rend difficile la concurrence car la production de conseils doit se faire de manière synchrone par rapport à l'application espionnée si l'on veut que les conseils produits soient cohérents avec l'état du système. Cet aspect constitue une des voies de recherche principales du projet.

4.5. Applications d'EpiTalk et projets en cours

L'architecture et le système EpiTalk sont maintenant dans une phase opérationnelle au LICEF, où plusieurs projets l'utilisant sont en cours. Une des directions privilégiées est celle de l'assistance au travail collaboratif et à distance (Giroux et al., 1996). L'architecture est utilisée pour la construction de plusieurs systèmes conseillers, en particulier pour le système *AGD*, un environnement de construction de cours (Paquette & Girard, 1996) qui connaît un succès grandissant (Paquette, 1996).

L'équipe de Pierre Marcenac à l'Université de la Réunion utilise EpiTalk pour des expérimentations sur les modèles de l'élève (Leman et al., 1995). Enfin, l'application de ces techniques aux interfaces homme-machine est prometteuse comme l'ont montré des expériences menées dans cette direction (Karsenty & Pachet, 1995).

5. Contraintes, objets et connaissances

Cette section décrit des travaux en cours sur la satisfaction de contraintes. Ces travaux sont réalisés avec l'aide de 2 étudiants, Pierre Roy et Anne Liret, en thèse sous ma direction.

5.1. Contexte

Le champ d'application des techniques de satisfaction de contraintes inclut de nombreux problèmes d'optimisation, ordonnancement, gestion de ressources, etc. Je me suis intéressé depuis quelques années plus particulièrement à la satisfaction *à domaines finis*, une branche de l'optimisation combinatoire dans laquelle les variables ne peuvent prendre qu'un nombre fini de valeurs, et connues d'avance. La CSP à domaines finis est, à notre avis, particulièrement bien adaptée à la PPO parce que les classes définissent naturellement des domaines discrets (on ne sait bien réifier que des concepts discrets, sauf des classes particulières comme les réels, les fractions ou les flottants, qui ont en général un statut exceptionnel dans le langage). D'autre part, les classes fournissent un langage (le langage des expressions constituées des messages définis dans les classes), dans lequel il est naturel d'exprimer des contraintes portant sur des objets.

Je vais d'abord situer ma démarche dans le contexte général des langages de contraintes, puis dans celui des contraintes dans les langages à objets.

1. *Satisfaction de contraintes et langage : trois sources d'inspiration*

Les premiers travaux ayant formalisé la notion de contrainte à domaines fini sont ceux de Macworth (Mackworth, 1977), qui a introduit les notions de base permettant le développement d'une algorithmique de la satisfaction de contraintes à domaines finis, en particulier la notion d'arc-cohérence. Aujourd'hui le mot contrainte est, comme le mot objet, terriblement polysémique. Outre les aspects purement algorithmiques dans la lignée des travaux de Macworth, l'orientation que je suis en ce qui concerne l'intégration de ces mécanismes dans un langage provient de trois sources : la Programmation Logique avec Contraintes, l'intelligence artificielle, et la Programmation par Objets.

1) Programmation logique avec contraintes

Les premiers langages ayant proposé d'intégrer des mécanismes de satisfaction de contraintes sont issus de la programmation logique. La programmation logique avec contraintes (CLP) est en effet une extension naturelle de la programmation logique, dans laquelle les variables logiques prennent leurs valeurs dans des domaines spécifiques (réels, domaines finis, booléens, etc.) (Codognet, 1995). Un des intérêts de voir les contraintes de ce point de vue est de bénéficier d'un cadre sémantique clair. De nombreux résultats d'efficacité ont été par ailleurs obtenus (Clp(fd) par exemple (Codognet & Diaz, 1996)). Un des reproches que l'on peut faire à cette famille de langages, dans notre contexte, est que la programmation logique se prête mal à une structuration des entités du domaine : les objets s'y sentent mal à l'aise.

2) intelligence artificielle et raisonnement formel

Un des premiers systèmes de satisfaction de contraintes à domaines finis est le système Alice de Jean-Louis Laurière (Laurière, 1976). Alice proposait une combinaison de mécanismes de satisfaction comme on les étudie aujourd'hui (essentiellement du filtrage de contrainte), avec des capacités de raisonnement formel. Du point de vue du langage, Alice proposait de plus un langage parfaitement déclaratif, basé sur la théorie des ensembles, et se démarquait ainsi des écoles traditionnelles de programmation (fonctionnelle ou logique). Un des points remarquables d'Alice est sa capacité à combiner du raisonnement formel avec un mécanisme de recherche brute (il est encore aujourd'hui un des seuls systèmes à proposer une telle intégration). Les travaux de Laurière ont montré que, dans certains cas, un raisonnement formel assez simple permettait de réduire énormément l'espace de recherche en trouvant directement des valeurs de variables ou en créant dynamiquement des contraintes plus simples au fur et à mesure de l'instantiation du problème. Un des défauts d'Alice est le fait qu'il ne manipule que des domaines plats : le langage d'expression des contraintes et essentiellement le langage arithmétique et ensembliste. Par ailleurs Alice est un système complexe, encore mal compris et donc difficile à évaluer (notamment, les heuristiques y jouent un rôle prépondérant).

3) La propagation de contraintes et les objets

Les travaux d'Alan Borning ont montré qu'il était possible de combiner avantageusement des mécanismes de propagation de contraintes avec les langages objets, et son système pionnier *ThingLab* (Borning, 1981) a donné lieu à toute une école de recherche dans ce domaine (le système *Kaleidoscope* (Freeman-Benson, 1990), *RollIt* (Karsenty & Weikart, 1994), *OTI Constraint Solver* (Borning & Freeman-Benson, 1995)). ThingLab est le premier système, à notre connaissance, à avoir exploité les structures objets dans le cadre de système de contraintes. Cependant, l'application de ces techniques à l'optimisation combinatoire est problématique : ces mécanismes relèvent de la propagation plutôt que de la satisfaction (Cf 5.1.2), ce qui limite leur champ d'application à des systèmes réactifs (interfaces graphiques typiquement).

Ces trois sources de travaux nous ont inspirés à des titres divers : problématique de *l'intégration* dans un langage de programmation (CLP), intégration de domaines *structurés* dans les contraintes (travaux de Borning), et soucis *d'adaptation* au domaine, issus des travaux de Laurière.

2. CSP et objets

Plusieurs systèmes proposent aujourd'hui d'intégrer des mécanismes de CSP avec des objets. On a traditionnellement classé ces systèmes en deux catégories (Borning et al., 1992)), en fonction des algorithmes utilisés. D'une part le modèle à *perturbation*, dans lesquels le système a comme tâche de rétablir l'équilibre d'un ensemble d'objets après une perturbation (l'exemple typique est le convertisseur degré Celsius/degrés Fahrenheit). Cette catégorie contient essentiellement les systèmes dans la lignée de ThingLab (Kaleidoscope, RollIt, etc.). D'autre part le modèle à raffinement (*refinement* model), dans lequel le rôle du système est de calculer une solution, vue comme une affectation de valeurs à des variables, qui satisfasse les contraintes du problème. La plupart des langages de satisfaction de contraintes rentrent dans cette catégorie (IlogSolver (Puget, 94), Laure (Caseau, 1994) et d'autres (Gensel, 1995), (Kökeny, 1994)).

Cette distinction, à notre avis, n'est pas essentielle : même si les deux modèles ne sont pas compatibles, les mécanismes de satisfaction de contraintes, vus de près, font appel à des techniques très proches de celles de la propagation. On peut classer les systèmes à objets + contraintes autrement, en fonction de la manière dont ils intègrent les contraintes dans le langage. La question de l'intégration se pose alors par « à quoi correspondent les variables contraintes dans le paradigme objet ? ». Il y a, selon nous, deux réponses possibles :

1) Modèles à attribut. Les variables contraintes sont identifiées aux attributs d'objets. Cette approche réalise une intégration intime des contraintes avec les objets, mais pervertit le modèle objet, car elle introduit la notion d'objet « partiellement instancié ». C'est le modèle sous-jacent à la plupart des systèmes sur le marché, de ThingLab à IlogSolver.

2) Modèles à classe. Les variables contraintes sont identifiées aux classes, via leur domaine. Elles ne sont pas rattachées à des classes, et le modèle objet reste inchangé.

Nous avons choisi d'explorer systématiquement la deuxième possibilité. Nous avons montré que pour les problèmes mettant en œuvre des objets composites, cette deuxième approche permettait d'une part de réutiliser des classes existantes telles quelles (un de nos thèmes principaux !), et d'autre part des gains en efficacité considérables, dus à un meilleur compromis entre propagation et instantiation (voir 5.3).

5.2. L'adaptation au domaine

La CSP est un ensemble de techniques générales, pour résoudre des problèmes NP-difficiles. Une des voies que nous explorons pour réduire la complexité est de représenter des connaissances exploitables par le système, pour l'adapter au domaine.

Il convient de rappeler ici que le système Alice proposait une forme d'adaptation, mais non pas au domaine, mais à l'*instance* de problème traitée. Dans notre contexte, l'adaptation au domaine se fait naturellement par le biais de *frameworks* bien conçus pour laisser des degrés de liberté, et représenter certains types de connaissances. L'identification des degrés de liberté et leur exploitation pour l'adaptation au domaine fait l'objet de plusieurs recherches en cours, et de plusieurs résultats : en matière de contrôle des algorithmes de filtrage, en matière de conception de problèmes, et enfin en matière de raisonnement formel.

1. Le framework BackTalk

BackTalk (thèse de Pierre Roy) est un *framework* pour l'intégration de mécanismes de satisfaction de contraintes à domaines finis en BackTalk, suivant l'approche à classe. BackTalk est construit en vue de faciliter l'expression de connaissances du domaine, exploitable par les algorithmes de CSP.

Pour ce faire, BackTalk repose sur une réification systématique des concepts importants pour définir et traiter un problème de contrainte : variables, domaines, contraintes, algorithmes et problèmes. D'une part, les domaines sont représentés par des classes, qui définissent donc un langage pour exprimer les contraintes. Ce langage est plus riche que le langage arithmétique habituel, et adapté au domaine, par définition, puisque les contraintes s'expriment par des messages définis dans les classes des objets, et non pas à travers un langage de contraintes général. Par ailleurs, les contraintes sont elles aussi représentées par des classes, qui définissent les méthodes de filtrage. Ces méthodes de filtrage sont essentielles pour assurer une efficacité raisonnable à un problème de contraintes. Comme ces classes de contraintes ne définissent *que* les méthodes de filtrage, on peut même affirmer que la sémantique des contraintes est entièrement déterminée - en BackTalk - par la donnée de ces méthodes. Enfin, BackTalk réifie aussi les algorithmes de résolution (*forward-checking*, *full look ahead*, etc.), qui sont organisés en une hiérarchie de classes.

BackTalk est un système efficace et bien intégré au langage Smalltalk (Roy & Pachet, 1997a; Roy & Pachet, 1997c). L'aspect minimal du système le rend particulièrement bien adapté à des fins pédagogiques : il a servi de base à trois tutoriels de conférences (Roy & Pachet, 1995; Roy & Pachet, 1996b; Roy & Pachet, 1997d).

2. Connaissances aux limites : le cas des mots croisés

L'exploitation effective des mécanismes de satisfaction de contraintes pose toute une série de problèmes difficiles de représentation de connaissances. Une des questions-clés est en particulier de trouver des représentation de connaissances du domaine qui permettent d'accélérer les résolutions, et donc qui soient exploitables par les techniques de CSP traditionnelles. Une application nous a permis d'obtenir déjà des résultats intéressants, dans un autre domaine : la génération de mots croisés. Le système *CrossTalk* permet de trouver des grilles de mots croisés à partir d'une grille dessinée à l'aide d'un éditeur, et d'une liste de mots arbitraire (voir Figure 7). L'ordre de grandeur d'une liste de mots raisonnable étant de 100.000, un mot croisé moyen de 10 sur 10 comportant environ 30 variables mots, on obtient un espace de recherche dont la taille est de l'ordre de 5000^{30} environ, soit 10^{110}. Outre l'utilisation de techniques classiques des CSP, l'idée ici est de représenter des connaissances sur la distribution des lettres dans les mots de manière à éviter des recherches inutiles. Par exemple, en français, la lettre « q » est habituellement suivie d'un « u » (sauf exceptions rares). Nous avons montré que ce type de connaissances peut avantageusement être représenté en BackTalk par des sortes de règles de contrôle, qui s'intercalent entre les algorithmes de résolution et les méthodes de filtrage. Ces règles de contrôle permettent d'accélérer considérablement la résolution des mots croisés comme nous l'avons montré dans (Roy & Pachet, 1997a), (Roy & Pachet, 1996a).

La recherche de solutions réelles pose par ailleurs un certain nombre de problèmes difficiles. Les grilles « thématiques » dans lesquelles on cherche à optimiser le nombre de mots qui prennent leur valeur dans une sous-liste donnée (par exemple des noms de département). Un autre problème difficile est celui des mots « illégaux » que l'on trouve souvent mais en petit nombre dans les grilles réelles (par exemple les cheville, avec des définitions comme « en avance, hors concours, tiré à part, en marche », etc.). Ces mots illégaux ne se justifient, dans la réalité de la construction de grille humaine que dans la mesure où ils permettent par ailleurs d'obtenir une « belle grille », c'est à dire une grille avec peu de cases noires, ou selon tout autre critère (Perec, 1986).

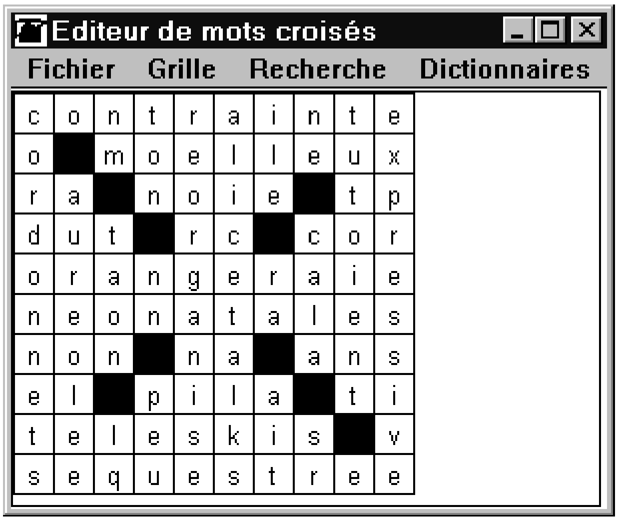


Figure 7. Une grille de mots croisés trouvée en quelques secondes par BackTalk/CrossTalk à partir d'une liste d'environ 150.000 mots français, du premier mot horizontal et sixième vertical imposés (les mots de 2 lettres sont ignorés).

5.3. Conception de problèmes objets + contraintes

Il est bien connu que la formulation d'un problème joue un rôle prépondérant dans sa résolution. Ceci est particulièrement vrai dans le cas des problèmes combinatoires. Suivant la représentation choisie, on peut, pour un même problème, exhiber une formulation qui conduit à un espace de recherche très grand, et une autre, équivalente, qui conduit à un espace beaucoup plus réduit (voir Laurière (Laurière, 1976), (Bourgine, 1989), (Amarel, 1966)). Ce phénomène existe déjà avec les CSP plats standards; avec des objets, il est encore plus frappant, car on multiplie la diversité des espaces problèmes possibles, avec la diversité des représentations objets équivalentes pour un même espace.

Nous avons travaillé sur la formulation de problèmes de CSP à domaines finis portant sur des objets composites, dans lesquels il existe à la fois des contraintes inter objet et intra-objets, à tous les niveaux d'une hiérarchie de partie. Nous avons obtenu un résultat spectaculaire en complexité pour cette classe de problèmes, en proposant une formulation beaucoup plus avantageuse dans laquelle :

1) On sépare la résolution du problème en plusieurs sous problèmes portant sur chaque niveau de composition,

2) On remplace les propagations des contraintes inter objet composite, par l'instantiation explicites d'objets composites, sur les lesquels ne portent plus que les contraintes entre objets composites.

Nous avons montré que cette approche permet des gains en complexité (un facteur de racine carrée) qui sont d'autant plus intéressants que les objets composites sont rares, par rapport au nombre total de n-uplets d'objets composés. Nous avons par ailleurs vérifié ce résultat expérimentalement sur un problème jouet (trouver des paires de rectangles satisfaisant certaines contraintes inter rectangle, et entre rectangles), puis sur un problème complexe dans le domaine musical : l'harmonisation automatique (Pachet & Roy, 1995a; Pachet & Roy, 1995b), (Roy &

Pachet, 1997b) (voir section 7.2.1). Dans ce dernier cas, nous obtenons des résultats meilleurs d'un facteur 10 par rapport à toutes les approches précédentes.

5.4. Travaux en cours et perspectives

Le *framework* BackTalk est aujourd'hui dans un état stable et obtient de bonnes performances en comparaison avec les autres systèmes équivalents du marché. Son architecture modulaire et extensible le rend particulièrement adéquat l'expérimentation et le prototypage rapide, et il sert aujourd'hui à plusieurs travaux en cours pour étudier divers aspects de la CSP et son intégration avec les objets.

1. Contraintes et hiérarchies de parties

Nous continuons nos travaux sur les problèmes de contraintes impliquant des hiérarchies de parties, pour généraliser nos premiers résultats. Un problème sur lequel nous tentons d'appliquer notre méthodologie est le problème de la reconstitution d'images à partir de certaines projections (polyominos (Chang, 1971)). Notre méthodologie de conception peut s'appliquer dans ce cadre en considérant des espaces problèmes constitués non pas des points de l'image, mais de formes géométriques calculées à l'avance, une fois pour toutes (dans le cas des images convexes, on considère l'espace constitué des *barres* verticales et horizontales formant les images). Nous avons déjà obtenu de très bonnes performances sur de premières expérimentations, et continuons dans cet axe. Nous essayons en particulier de combiner notre approche avec des algorithmes SAT spécialisés sur ce problème, développés par Olivier Dubois du Lip6.

2. CSP et raisonnement formel

Le projet AliceTalks (Liret et al., 1996) a consisté en une réécriture critique du système Alice de Jean-Louis Laurière (Laurière, 1976). Ce travail a comme but d'explorer certaines des caractéristiques du système Alice à la lumière des techniques modernes de satisfaction de contraintes. En particulier, la capacité d'Alice à combiner raisonnement formel et énumération lui permet d'obtenir des performances dans certains cas hors de portée des techniques actuelles. Le postulat de base que nous faisons est que ces mécanismes doivent être contrôlés de manière très fine par des connaissances du domaine, et pas simplement par des heuristiques générales comme dans le système Alice original.

La thèse de Anne Liret sous ma direction devrait apporter des éléments de réponse à ce sujet. Cette thèse étudie l'utilisation de techniques de réécriture pour le raisonnement formel. Cette capacité de raisonnement devraient permettre d'exploiter des connaissances sur le domaine fournies à l'avance (par exemple sous forme de bases de règles logiques), afin de faire des inférences permettant de réduire l'exploration de l'espace problème. In fine, le but est d'intégrer de telles capacités en BackTalk pour proposer un système complet et facilement adaptable.

3. Contraintes temporelles

Nous poursuivons des recherches pour l'application de notre démarche au cas des contraintes temporelles (TCSP, (Dechter et al., 1991)), un formalisme développé pour traiter explicitement des contraintes portant sur des intervalles de temps. Les applications de ces techniques sont nombreuses par exemple dans le domaine médical, ou nous poursuivons une collaboration avec Michel Dojat sur les problèmes de reconnaissances de scénarios (Ramaux et al., 1997). Un des objectifs de ce travail est d'intégrer les techniques particulières développées pour les TCSP dans le cadre de BackTalk, de manière à bénéficier d’un système capable de gérer les deux types de contraintes indifféremment. A plus long terme, un des axes poursuivis est la formalisation et le traitement des structures temporelles *circulaires*, travail déjà amorcé dans le contexte de problèmes d’analyse musicale (Pachet & Carrive, 1996).

6. Sens commun

J'ai effectué un stage post-doctoral au département d'informatique de l'Université du Québec à Montréal, sous la direction d'Hafedh Mili. L'équipe de H. Mili s'intéresse depuis longtemps à la modélisation de réseaux hiérarchiques, et a développé un modèle de hiérarchies sémantiques basé sur la *régularité*, une généralisation de l'héritage (Mili, 1988). Par ailleurs, cette équipe avait acquis une copie de la base de connaissances Cyc (Lenat & Guha, 1990) dans le but 1) d'utiliser le réseau sémantique sous-jacent à Cyc pour aider à la génération de textes, 2) de tester une hypothèse de régularité émise sur les documents argumentatifs. Cette hypothèse, si elle se vérifie, permettrait d'engendrer automatiquement un certain nombre de documents argumentatifs à partir de squelettes de raisonnements d'une part (une structure logique), et de bouts de textes organisés préalablement en un réseau sémantique.

Un des enjeux du stage était d'opérer effectivement la liaison entre les mécanismes et modèles de la régularité et le système Cyc. Malheureusement, l'état du système Cyc à l'époque, et le manque de temps m'ont empêché de terminer ce projet. De plus, la taille gigantesque de Cyc a poussé ses concepteurs à effectuer des optimisations d'implantation qui le rendait peu adapté aux explorations logiques profondes requises par la génération de textes. Néanmoins, l'étude des patrons de régularité dans Cyc nous a amené à généraliser la notion de régularité proposée par Mili, et à formuler un certain nombre d'hypothèses quant à la structure logique de la base de connaissances.

6.1. Cyc

Le système Cyc, développé par l'équipe de Douglas Lenat à MCC, est un système ambitieux, tentant de représenter la connaissance de «sens commun»[6]. Par sens commun, Lenat entend les connaissances en général non-explicitées, mais nécessaires à la compréhension de dictionnaires, d'encyclopédies, d'articles du Wall Street Journal, ou de France-Soir, mais aussi les connaissances qui nous permettent de ne pas croire ce qu'on lit dans Allo-Police, ou le *tabloid* sensationnel américain *National Enquirer*. On y trouve, ou on est capable d'y prouver, des règles comme «des objets tombent quand on les lâche», «des enfants sont moins âgés que leurs parents», «arracher un membre fait souffrir», «les êtres humains veulent être heureux», ou bien encore «après un acte de vente, l'acheteur possède l'objet acheté». Le projet CYC, étalé sur 10 ans, a commencé en 1984. Peu de documents sont à ce jour disponibles sur l'état effectif du système, à part le livre (Lenat & Guha, 1990) qui décrit les principaux mécanismes de représentation et d'inférence, ainsi que le rapport de mi-parcours (Lenat et al., 1990), qui donne une idée de la complexité du système. Cyc est basée sur l'hypothèse que l'intelligence ne peut exister sans une masse critique de connaissances, que Lenat se propose de construire manuellement.

Ces hypothèses, discutées en détail dans (Lenat & Feigenbaum, 1991), ont été vivement critiquées, en particulier dans le même numéro du AI Journal (Cantwell Smith, 1991). Le système fait l'objet de nombreux débats dans la communauté d'intelligence artificielle. Le livre sur Cyc a aussi fait l'objet de critiques vives - et tardives - (CycBookReviews, 1993) auxquelles ont répondu les auteurs avec un point de vue d'ingénieur. Signalons enfin le livre (Steels & McDermott, 1994)

[6] Même remarque pour sens commun comme traduction littérale de *common sense* (Cf. note 9).

(chapitre 4) transcrivant un atelier au cours duquel Lenat eut à défendre ses idées devant un public peu facile. Si le débat est le plus souvent passionnant, les problèmes techniques restent, là encore, peu abordés et les débats glissent rapidement vers des questions philosophiques touchant à la nature de la connaissance, ou la définition de l'intelligence.

D'un point de vue technique, les descriptions initiales de Cyc étaient fortement orientées vers les langages de frames. Celles-ci ont maintenant disparu, au moins superficiellement, et Cyc est aujourd'hui décrit de manière entièrement logique, i.e. en terme d'assertions, portant sur des constantes, par des prédicats ou relations (Guha & Lenat, 1994). Bien qu'il soit difficile de mesurer la taille de la base, celle-ci est estimée à environ deux millions d'assertions (Cf. réponse de Guha et Lenat dans (CycBookReviews, 1993)), mentionnant environ 50,000 «constantes» et 8,000 «collections», à l'aide de 5,000 «prédicats». Ces assertions décrivent les catégories d'objets les plus fréquentes des objets concrets, tangibles, courants (chaises, lacs et voitures) aux objets abstraits, intangibles, comme les maladies, les transactions financières, et les différents buts et croyances des agents.

6.2. Régularité dans Cyc

Un des travaux effectué lors de mon stage post-doctoral consistait à trouver des moyens de réduire la complexité du système, en exploitant la notion de régularité développée par l'équipe de Mili. La notion de régularité est une sorte d'isomorphisme d'arbres, qui permet de parler d'une relation (un slot) d'une manière abstraite. Une relation est dite régulière si elle vérifie une propriété selon laquelle l'image de tout nœud d'une hiérarchie est toujours incluse dans l'image de tout super nœud dans la hiérarchie de départ.

Prenons comme exemple le concept `Automobile` représentant la collection des automobiles, et ses différentes spécialisations, dont `FrenchCar`, elle-même spécialisée en `PeugeotCar` laquelle est spécialisée en `Peugeot403Car`, et `SouthKoreanCar`, spécialisée en `HyundaiCar`. Cyc propose le slot `ManufacturingOrganization`, qui associe à toute instance `d'Automobile`, le constructeur automobile correspondant, qui doit être une instance de `AutomobileManufacturer`. De même, Cyc associe à chaque constructeur une zone géopolitique (instance de `Region`) qui correspond à la base de fabrication, ou, la «nationalité» du constructeur, par le slot `RegionOfMainActivity`. La figure 2 montre les hiérarchies (`Automobile, is-a`) et (`Region, sub-region`). La correspondance entre les deux hiérarchies pourrait être exprimée par la régularité d'une propriété que l'on exprimerait par `RegionOfMainActivity-Of-ManufactoringOrganization`, ou, la concaténation de ces deux relations.

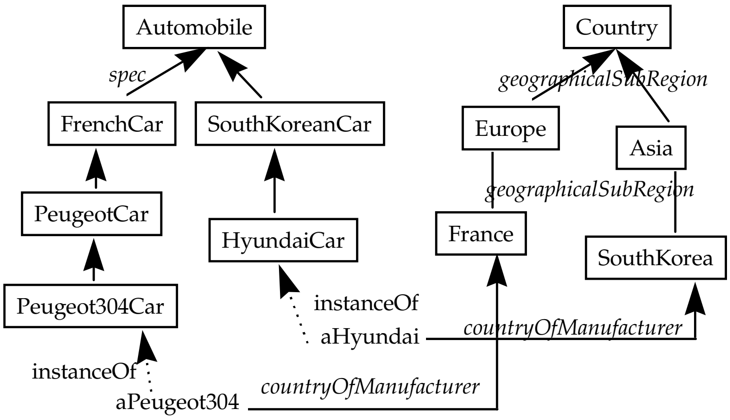


Figure 8. Régularité d'une relation entre deux hiérarchies de classes en Cyc. Ici, `countryOfManufacturer` est régulière par rapport a `spec/geographicalSubRegion`.

Nous avons généralisé la notion de régularité proposée initialement sur des hiérarchies simples en substituant les propriétés binaires par une notion plus générale de chemin d'accès (Pachet & Mili, 1994). Nous avons par ailleurs proposé d'utiliser cette relation pour aider les utilisateurs à naviguer dans des bases de données hypertextuelles dans (Mili & Pachet, 1995a; Mili & Pachet, 1995b).

6.3. Utiliser Cyc

L'expérience Cyc constitue une expérience singulière dans le contexte de nos autres travaux. Une des hypothèses fortes de D. Lenat dans ce projet est de postuler l'existence d'une couche de représentation (le sens commun), qui soit représentable de manière indépendante de toute utilisation. Cette hypothèse ne fait pas l'unanimité dans les milieux de l'IA pour ce qui est des domaines dits "métiers" : deux experts de deux domaines différents ont des définitions différentes pour le même concept (par exemple, l'eau vue par un chimiste, un biologiste, un physicien, un nutritionniste, etc.). Pour cette couche particulière, qui par définition fait consensus, elle est, à notre avis, tout-à-fait raisonnable.

En revanche, le projet Cyc soulève la question de la possibilité non pas de construire, mais *d'utiliser* des bases de connaissances de grosse taille. En effet, l'expérience d'utilisation de Cyc à fait surgir un effet paradoxal : plus la base est grosse, plus il faut de connaissances pour l'utiliser, et ce, de manière exponentielle. Dans le cas des inférences de sens commun, il ne suffit pas de connaître un vocabulaire particulier (comment exprimer l'idée de "posséder"), mais aussi les règles qu'il contient. Ainsi, la différence que le système entretient entre "owns", "possess", "has" est-elle uniquement représentée par le fait que le système contient des règles d'inférence différentes concernant chacun de ces termes, et que donc, les inférences que le système sera capable de faire seront différentes. Si la quantité de connaissances nécessaire à utiliser la base de manière productive est effectivement exponentielle, alors le système est inutilisable. Ceci prouverait que les connaissances de sens commun, si elles sont rationnelles dans leur mécanique et leur logique interne, ne le seraient pas dans la manière dont on les utilise. Même si le projet Cyc semble aujourd'hui ne pas avoir abouti à un succès, la question de l'existence d'une telle couche de connaissance reste, à notre avis, ouverte.

7. Musique et objets

Cette section est consacrée aux travaux que j'ai effectués dans le domaine de la musique et de l'intelligence artificielle. J'ai développé ou participé au développement d'un certain nombre d'applications musicales complexes, en utilisant la plupart des mécanismes de représentation que décrits dans les sections précédentes.

7.1. Musique et informatique

Le rapprochement entre intelligence artificielle et musique a des racines profondes. La musique, au contraire des autres arts, a depuis longtemps été l'objet de formalisations et de théories qui se prêtent directement à la modélisation informatique, et ce, depuis l'apparition des premiers traités d'harmonie et de contrepoint.

Mon approche de la musique consiste à la prendre comme *prétexte* pour étudier certains types de connaissances, en particulier analytiques. La musique tonale fournit alors un certain nombre de *problèmes bien posés*, sur lesquels on peut construire des modèles, les implémenter et expérimenter des idées. Les modèles ainsi construits, les solutions proposées, n'ont ici d'intérêt que lorsqu'ils dépassent le cadre de la musique, et sont applicables dans d'autres champs. Cette approche n'est bien sûr pas la seule possible (voir par exemple le dossier IA et Musique en France, (Pachet & Assayag, 1995) et l'article de synthèse (Roads, 1985)); elle est même minoritaire au sein de la communauté d'Informatique Musicale, mais elle est importante car elle permet de donner un but rationnel et des moyens de validation, voire une véritable assise "scientifique" à l'Informatique Musicale que d'autres problématiques tout aussi passionnantes - comme la composition assistée par ordinateur - ne peuvent pas avoir.

D'un strict point de vue héréditaire, ce rapprochement musique et IA n'est pas récent. Outre la communauté active de l'Informatique Musicale, de nombreux chercheurs sont simplement "passés" par la musique pour nourrir des réflexions en IA et programmation[7]. Aux Etats-Unis, Douglas Hofstadter s'est rendu célèbre par son livre "Gödel, Escher Bach" (Hofstadter, 1985) qui mettait en relation des œuvres musicales, des dessins et des théorèmes. La musique continue d'être une source d'inspiration pour ses travaux plus récents, comme le projet CopyCat où il exploite l'analogie entre *patterns* mélodiques et séquences de nombres (Hofstadter, 1995). Marvin Minsky a écrit un texte sur le sens commun[8] et la musique, ou il évoque entre autres le problème primordial de la représentation des mélodies (Minsky & Laske, 1992). Il a, entre autres, introduit un modèle du raisonnement vu comme une société d'agents communicants, avec des applications directes à la modélisation de processus musicaux analytiques (Minsky, 1985). Sa théorie des "K-lines" a par ailleurs des applications directes à la musique (Horowitz, 1995). Toujours au MIT, Henry Lieberman s'est intéressé aux éditeurs de partitions, aux synthétiseurs, tout au long de ses recherches sur les interfaces utilisateurs et les environnements de programmation (Liebermann, 1991). Luc Steels a montré comment les mécanismes de propagation de contraintes pouvaient

---

[7] Le chemin parcouru par ces chercheurs mériterait une étude particulière sur la manière dont certaines préoccupations musicales se généralisent à des problèmes d'intelligence artificielle.

[8] Sens commun est ici à comprendre comme traduction littérale, faute de mieux, de l'anglais *common sense*.

résoudre le problème de l'accord de passage (*passing chord problem*) (Steels, 1979), avant de s'intéresser, entre autres, à l'acquisition de connaissances et aux software agents. Sur un plan plus technique, mais tout aussi intéressant, de nombreux chercheurs en programmation par objets se sont intéressé à la musique (voir (Pachet, 1997b) pour une synthèse plus complète). Ralph Johnson a participé au développement de l'environnement de synthèse sonore KYMA avec Carla Scaletti (Scaletti & Johnson, 1988), puis a travaillé sur un compilateur Smalltalk optimiseur, avant d'étudier des problèmes de conception objets (les *frameworks* et les *patterns (Gamma et al., 1994)*). Dans la même communauté, Steven Pope a développé un environnement de développement d'outils musicaux en Smalltalk (Pope, 1991), il est par ailleurs un des acteurs de l'architecture MVC, et auteur du fameux "Cookbook MVC" (Krasner & Pope, 1988).

Ce phénomène n'est pas uniquement américain. En France, Patrick Greussay, passionné par les variations Diabelli, a proposé un modèle - les "graphes Beethovéniens" - rendant compte de son analyse, personnelle, de cette œuvre réputée difficile (Greussay, 1973) (Greussay, 1985). Ses travaux sont à l'origine de toute une école de recherche en programmation fonctionnelle. Dans cette lignée, Pierre Cointe a travaillé sur un langage de composition assistée par ordinateur permettant de représenter et manipuler des processus musicaux : le langage *Formes* (Rodet & Cointe, 1991). Il s'est par ailleurs intéressé à l'étude systématique des langages à objets, de leur implantation et en particulier leur caractère réflexif (le langage *ObjVlisp*, (Cointe, 1984)). Jean-Pierre Briot a suivi un parcours parallèle, travaillant d'abord sur Formes, et passant ensuite à l'étude des métaclasses, puis des langages d'acteurs (Briot, 1989b). Plus récemment, Francis Rousseaux a proposé d'intégrer des mécanismes d'apprentissage symbolique pour l'enseignement du solfège avec le logiciel *Le Musicologue* (Rousseaux, 1990), et s'est ensuite intéressé à la modélisation des situations de crise (Rousseaux, 1995).

Bien d'autres chercheurs célèbres, en particulier en linguistique computationnelle, comme Winograd, Smoliar (voir par exemple le numéro spécial de AI Journal sur la musique (Smoliar, 1995)), Steedman, etc. ont parcouru des chemins similaires, passant par la musique pour poser des problèmes, puis trouver des solutions qui intéressent toute la communauté d'IA.

7.2. Origines de MusES

J'ai effectué mon stage de DEA au département pédagogie de l'Ircam en 1987, sous la direction de David Wessel et Jean-Louis Laurière. Le cadre de travail de l'Ircam était passionnant pour quiconque avait une double formation et fascination pour Lisp et pour la norme MIDI, alors en pleine heure de gloire : c'était le début du langage MIDI-Lisp (Boynton et al., 1986) puis plus tard preFORM, sorte de plate-forme devant permettre aux musiciens non informaticiens de fabriquer eux-mêmes leur programmes. Mon stage de DEA m'a permis de travailler sur deux problèmes de représentation de connaissances musicales : l'analyse harmonique automatique et la simulation d'improvisation (Pachet, 1987a; Pachet, 1987b).

1. *Analyse harmonique automatique*

Le problème de l'analyse harmonique consiste à produire une interprétation de la tonalité pour chaque accord d'une séquence d'accords. Ce problème musical a été beaucoup étudié par les musicologues et les informaticiens (voir (Pachet, 1997a) pour une synthèse). La motivation pour le système d'analyse était double. D'une part il était nécessaire d'avoir un mécanisme d'analyse automatique pour traiter les séquences d'accords en vue d'obtenir les tonalités, nécessaires à l'improvisation. D'autre part, j'avais lu le papier de Steedman sur les grammaires de grilles de Blues (Steedman, 1984) qui m'avait à la fois fortement impressionné, et terriblement déçu. Ce papier décrit en effet une grammaire très compacte (8 règles) qui rend compte de manière parfaite de l'essence des grilles de Blues. Malheureusement, la présence de règles dépendantes du contexte la rend inutilisable pour un système opérationnel, et ce, pour des raisons théoriques profondes (on ne sait pas fabriquer des analyseurs pour ces grammaires). Le système d'analyse que j'ai produit permettait de calculer *effectivement* les tonalités de grilles de Jazz courantes. Son principe reposait sur une vision de l'analyse comme un processus de reconnaissance en deux étapes : d'abord on identifiait des "zones" connues, correspondant à des structures musicales claires (2-5-1, cadences, etc.) dont la tonalité était évidente. Ensuite on procédait à des regroupements de ces formes de base, grâce à des règles de harmoniques simples. Par exemple, un accord isolé pouvait se faire agréger par une forme reconnue s'il était analysable dans la tonalité de la forme en question, etc. (Pachet, 1991b).

Le premier système ainsi élaboré souffrait cependant de nombreux défauts. La difficulté que j'avais à améliorer mon système m'a poussé à m'intéresser de plus près au *langage* utilisé pour représenter ces connaissances. Le langage initialement utilisé était une combinaison de lisp, de langage objet non standard (preForm), et de moteur d'inférence (l'Experkit, (Ferber, 1987)). Ce langage avait comme principal intérêt de proposer une intégration de structures objets dans les règles. Mais cette intégration était une couche rajoutée au dessus d'un mécanisme qui n'était pas conçu pour ça. L'utilisation de ce langage et l'analyse de ses limites m'ont incité ensuite à travailler dans la direction de l'intégration d'objets et des règles, avec un autre point de vue (objets Smalltalk au lieu des "frames" d'ExperKit), et règles à la OPS-5 au lieu des règles à la Snark), d'ou mes travaux sur NéOpus.

J'ai pu retravailler sur le système d'analyse plus tard, notamment en partant des résultats de la thèse de Michel Dojat, qui a dégagé un cadre pour la représentation de raisonnements temporels (Dojat, 1994). La principale application de ce cadre est le système NéoGanesh, réalisé à l'hôpital Henri Mondor. L'idée de base de Michel Dojat est de fonder les raisonnement temporels sur deux mécanismes de base : agrégation et oubli (Dojat & Sayettat, 1996). Faisant écho à mes propres travaux sur le système d'analyse, les travaux de Michel Dojat m'ont permis de progresser dans mes réflexions, et de proposer une nouvelle version du système d'analyse harmonique automatique. Partant de la représentation des objets musicaux temporels de MusES (Cf. 7.4), j'ai reconstruit le système d'analyse harmonique automatique dans ce cadre. Cette reconstruction m'a permit de dégager un point important que je n'avais pas remarqué lors du premier prototype : l'importance du modèle temporel. Ma première vision du problème de l'analyse harmonique était essentiellement atemporelle : l'analyse s'effectue en temps différé (pas de contraintes de temps), je considérais en effet qu'un modèle rendant compte simplement de la juxtaposition des accords était suffisant. En fait, le modèle du temps est ici beaucoup plus important, comme l'ont montré mes travaux effectués avec Jean Carrive, en particulier, sur la formalisation des intervalles temporels circulaires (Pachet & Carrive, 1996). Ce dernier travail montre que la prise en compte de la circularité des séquences d'accords de Jazz est essentielle, et non pas fortuite, et que seul un modèle du temps adéquat permet de rendre compte de cette particularité du domaine. Le système reconstruit sur cette base acquiert ainsi un statut plus clair (Pachet, 1997a), (Mouton & Pachet, 1995), et il obtient de très bonnes performances.

2. *Simulation d'improvisation*

Le second problème - la simulation d'improvisation de Jazz - était un sujet à l'époque encore non traité, du moins dans la littérature disponible. Une des premières tâches fut de délimiter le style et la problématique étudiée. J'ai proposé un modèle permettant de simuler un bassiste accompagnateur de musique improvisée dans le style Be-bop, et respectant les contraintes suivantes : 1) pas d'utilisation de mécanismes aléatoires ni de calculs numériques et 2) possibilité de donner au système des connaissances expertes, issues de ma propre expérience, dans un formalisme compréhensible.

Les raisons justifiant ces contraintes sont évidentes : il s'agissait d'aborder le problème difficile de la compréhension du processus d'improvisation, à travers un modèle computationnel clair, et par l'expérimentation, avec un système facilement modifiable et compréhensible. Il s'agissait d'une certaine manière d'une contradiction dans les termes, puisque l'idée de base était de faire un système le plus transparent possible, mais qui soit capable de générer une musique non automatique, voire même originale, et en tout cas surprenante.

En pratique, il s'agissait en particulier de fournir un cadre conceptuel permettant de représenter un certain nombre de connaissances musicales du musicien de Jazz (moi-même en l'occurrence), comme la règle "on joue, en général, la tonique sur le premier temps", ou "on peut, pour passer d'une tonique à une autre, jouer un arpège ascendant suivi d'un chromatisme".

J'ai proposé pour ce faire une architecture pour un système de simulation, reposant sur la notion de "stratégie", ainsi que sur un mécanisme d'inférence à base de règles de production organisées en plusieurs niveaux (Pachet, 1991f). Les stratégies étaient des réifications des actions musicales rudimentaires (outre un indicateur appelé "niveau", représentant le *degré d'abstraction* de la stratégie), mais suffisantes pour pouvoir exprimer les règles typiques du bassiste. Les exemples typiques de stratégies sont les actions suivantes : jouer une pédale sur une note donnée ; jouer un pattern rythmique intéressant ; jouer une séquence d'accords intéressante à transposition variable ; adopter une direction mélodique (monter ou descendre) ; ne pas jouer, etc.

Cette dualité entre stratégies (abstraites), et notes (concrètes) était fortement inspirée du système de jeu d'échecs *Robin* de Pitrat (Pitrat, 1977). Partant de l'inadéquation fondamentale des mécanismes basés sur des arborescences de "coups" (alpha-béta), Robin développe une arborescence de plans, bien plus réduite, et qui exploite une grande quantité de connaissances expertes (au contraire de l'arborescence des coups, qui nécessite une fonction d'évaluation difficile à produire). Même si aujourd'hui la puissance des machines rend l'étude du jeu d'échecs moins urgente, l'idée de Pitrat est une idée moderne : c'est en trouvant des réifications d'objets abstraits bien choisis que l'on pourra exprimer des connaissances utilisables par la machine.

Ce projet a donné lieu a la réalisation d'un programme simulant le bassiste, qui n'est aujourd'hui plus disponible : il tournait sur une version intermédiaire de Le_lisp, avec une version non finale de l'ExperKit et de MidiLisp, spécifiques au Macintosh Plus, logiciels aujourd'hui disparus. Néanmoins la conclusion de ce stage fut multiple :

- Les expérimentations réalisées ont permis de valider l'architecture proposée par un système qui ne pouvait pas engendrer de musique (MIDI), mais arrivait à afficher les notes produites, et ce, en temps réel. La puissance des machines disponibles à l'époque ne permettait pas d'expérimenter en temps réel avec le système complet (par manque de mémoire plus que de puissance de calcul).

- La mise en évidence d'un goulot d'étranglement pour la représentation de connaissances. En effet, si l'architecture choisie était, d'une certaine manière, construite pour pouvoir ingurgiter exactement les connaissances que je voulais y introduire, certains nœuds de l'architecture restaient "sans voix". Je faisais finalement le constat qu'il me manquait des connaissances, ou plutôt que les connaissances, en particulier stratégiques (choix de stratégies en cas de conflit), s'avéraient maladroites à expliciter, car ne correspondant plus à aucune intuition musicale reconnaissable.

- L'importance du temps réel, conduit à la conclusion que le manque de temps est le moteur principal de pensée musicale. Une des conclusions du rapport est que la règle la plus importante de la musique improvisée est : "si on a le temps, on réfléchit, sinon on joue plus ou moins n'importe quoi" (p. 45).

Mon système de simulation de bassiste jouant du Jazz était basé sur une réification judicieuse des actions musicales, mais je n'ai pas eu le temps de mettre à jour son implémentation avec les techniques « modernes » de représentation, ni de conduire les expérimentations nécessaires. Par ailleurs, cette architecture prévoyait un module de mémorisation sommaire, sous la forme de deux stratégies particulières (une pour « enregistrer», une autre pour « récupérer ») qui était insuffisant pour rendre compte de la complexité et du rôle prépondérant de la mémoire musicale dans l'acte d'improvisation.

Une extension de ce modèle primitif de la mémoire vers un modèle plus sophistiqué a été effectuée par Geber Ramalho au cours de sa thèse (Ramalho, 1996). Son modèle de la mémoire utilise les techniques de raisonnement à partir de cas. Le système résultant, *ImPact,* réutilise entre autres le système d'analyse harmonique réécrit en NéOpus[9]. Geber Ramalho a par ailleurs proposé un formalisme pour rendre compte de la combinaison des stratégies (renommées Pacts, comme *Potential Actions*) (Ramalho & Ganascia, 1994; Ramalho & Pachet, 1994). Les règles d'activation et de combinaison de Pacts sont écrites en NéOpus.

7.3. Le système MusES

Le système *MusES* est un effort pour représenter un certain savoir « consensuel » sur la musique tonale. MusES se présente sous la forme d'une bibliothèque de classes, représentant les concepts fondamentaux de cette musique : pitch-classes, notes, accords, gammes, mélodies. La taille de cette bibliothèque est d'environ 100 classes et 1500 méthodes (Pachet, 1994a; Pachet, 1994b). MusES n'est pas une application, mais plutôt une couche de base de représentation, utilisée pour construire des applications (voir Figure 9).

La réalisation même de cette bibliothèque pose une question philosophique de base : existe-t-il des représentations *universelles* de la notion de note musicale ? Idem pour les autres notions de base comme les accords, gammes, arpèges, etc. Si la réponse à cette question en général est clairement négative, on peut en revanche raisonnablement espérer construire des représentations universelles à l'intérieur d'une classe d'applications, ici caractérisée par des problèmes « à tendance analytique », sur des corpus de musique occidentale tonale, à tempérament égal.

Les premiers concepts à représenter en musique tonale sont ceux de note (indépendante de l'octave ou *pitch-class*), et d'altération (les dièses, bémols et autres signes, destinés à modifier la hauteur d'une note). Le problème est important : il s'agit de représenter non pas tant le signe lui-même, qui ne pose pas de problème, mais bien l'*intention* harmonique qu'il désigne. 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 intentions musicales bien différentes, aussi différentes que le sens des mots « sang », « sans », « s'en », ou « cent »[10].

---

[9] Il utilise une version modifiée, adaptée au problème de l'improvisation : cas concret de réutilisation de base de connaissances !

[10] 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 intention harmonique.

Le problème est alors que ces altérations forment 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. J'ai montré qu'une manière 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 étant des instances de classes différentes, suivant leur altération (Pachet, 1994b).

Une fois la théorie des pitch-classes 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 est de considérer les pitch-class comme des classes, au sens de la PPO. Mais ceci pose des problèmes non résolus : il faut un langage permettant de définir ses propres métaclasses, comme CLOS, ou ClassTalk (voir 3.2). La solution adoptée en MusES est l'agrégation : `OctaveDependentNote` est une classe qui agrège une *pitch-class* ET un numéro d'octave.

Les notions plus complexes sont alors ajoutées, comme les intervalles et leur typologie, qui doit respecter aussi 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 présentes, la limite de la bibliothèque étant fixée de manière arbitraire...

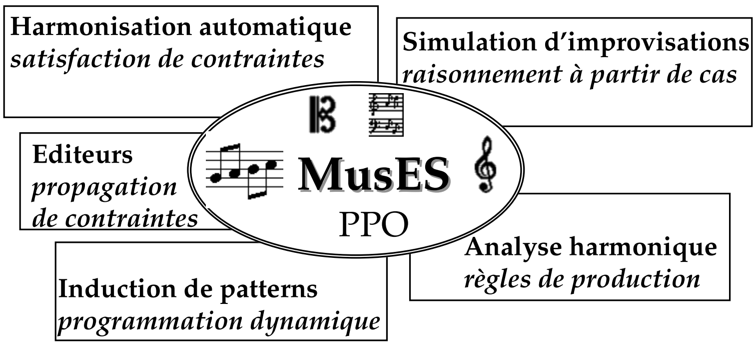


Figure 9. Le système MusES et ses principales applications.

Le système MusES est aujourd'hui utilisé dans plusieurs applications (voir (Pachet, 1997b) pour une synthèse). D'une part dans le système d'analyse harmonique, sur lequel j'ai retravaillé, en utilisant NéOpus (voir section 7.2.1). D'autre part, pour le système d'improvisation de Geber Ramalho (Ramalho, 1996). Pierre-Yves Rolland utilise MusES comme base pour étudier l'induction automatique de motifs dans les chorus de Jazz (Rolland & Ganascia, 1996). MusES est par ailleurs utilisé pour étudier les mécanismes de satisfaction de contraintes, sur le problème de l'harmonisation à quatre voix (voir 7.5). Enfin, une interface fondée sur la théorie des espaces harmoniques de Simon Holland (Holland, 1989) a été implémentée au dessus de MusES par des étudiants de DESS.

7.4. Un modèle du temps

Une fois le cas des objets « statiques » traité, se pose rapidement le problème de la représentation d'objets temporels. La représentation du temps pose de nombreux problèmes à la fois philosophiques (ontologies temporelles) et pratiques. De nombreux travaux en IA sont consacrés à la représentation du temps. On peut les regrouper en fonction du type de primitive temporelle sur laquelle ils reposent : *intervalle* (Allen, 1984), *points* (McDermott, 1982) ou *événements* (Kowalski & 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 aux manipulations musicales, et à l'idée même de note musicale.

Nous avons développé un *framework* pour la représentation des objets temporels musicaux en MusES. Ce *framework* repose sur une ontologie temporelle proche de celle de Michel Dojat pour son système de suivi ventilatoire. Ce *framework* est décrit dans (Pachet et al., 1996b; Pachet et al., 1995b). Nous allons ici simplement en décrire les points les plus importants, en le comparant avec celui du système MODE, réalisé par Steven Pope (Pope, 1991), et qui fait office de standard en matière d'environnement de développement d'applications musicales par objets.

On peut distinguer deux grandes manières de représenter les objets temporels, suivant le choix adopté pour représenter ces intervalles : 1) le temps est représenté hors des objets, dans des collections temporelles, c'est le choix du système MODE, et 2) le temps est représenté « dans » les objets eux-mêmes, c'est le choix adopté pour le système MusES.

1. *Temps hors objet*

Le système MODE propose une représentation du temps basée sur la notion d'`Event`, classe abstraite représentant des processus temporels. Sa représentation utilise le *design pattern Composite* de (Gamma et al., 1994). Cette classe possède une variable d'instance `duration`, représentant la durée du processus, dans une unité arbitraire, elle-même réifiée. Par ailleurs, les `Event` peuvent avoir un certain nombre de propriétés, accessibles par un dictionnaire. Le point important est que ces événements 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 bien déterminée. Ceci permet en pratique de les partager dans plusieurs collections temporelles, et de faciliter un certain nombre de taches d'édition (copier/coller). `Event` étant une classe abstraite, une hiérarchie de classes concrètes est construite pour représenter les événements courants comme les notes. La deuxième notion de base est celle d'`EventList`, qui représente une collection d'événements. Cette collection est constituée de paires (`duration`, `Event`), où `duration` représente le temps initial de l'événement en question, exprimé comme une durée par rapport au début de la liste. Cette représentation permet de dissocier le temps initial de l'objet temporel, et favorise ainsi la modularité. En outre, elle permet de représenter, récursivement, les listes temporelles comme des événements, à moindre frais : `EventList` est une sous-classe de `Event`. Ainsi, une `EventList` peut elle-même être considérée comme un objet temporel, ce qui permet de créer des structures temporelles arborescentes.

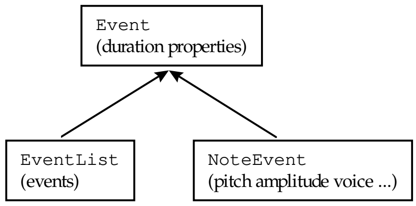


Figure 10. La hiérarchie des objets temporels en MODE.

2. *Temps dans les objets*

L'approche suivie dans le système MusES 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, et pour lesquelles la représentation précédente est alors maladroite. Ces objets temporels sont représentés à l'aide de trois notions de base :

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

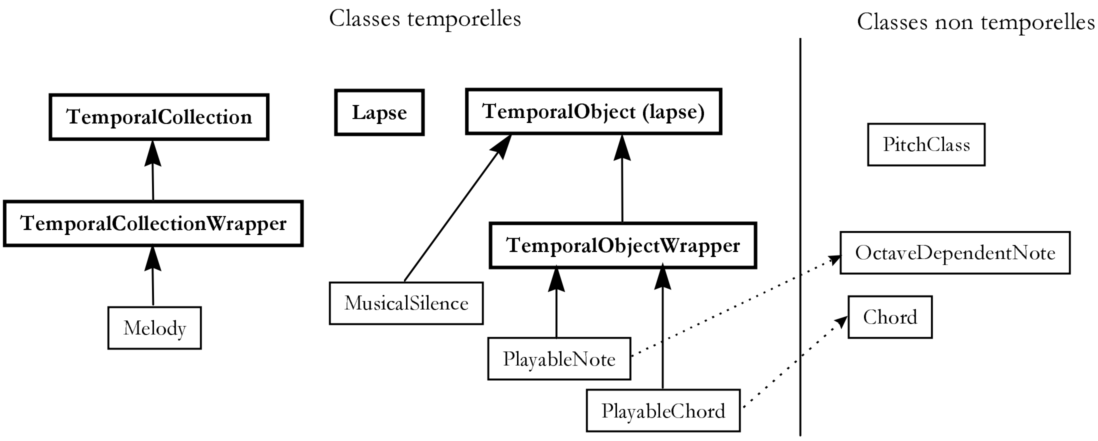


Figure 11. La hiérarchie des classes temporelles en MusES. Les flèches pleines représentent l'héritage de classe. Les traits pointillés la délégation.

Par ailleurs, MusES, comme nous l'avons vu plus haut, 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 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 de délégation (Liebermann, 1986), qui permet, entre autres, d'éviter l'emploi de l'héritage multiple. Nous avons introduit de plus une notion supplémentaire, permettant de mettre en oeuvre cette délégation de manière générique : celle d'*enveloppe temporelle* (`TemporalObjectWrapper`). Cet objet possède une variable d'instance 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 5 classes, à partir desquelles tous les objets temporels de MusES sont fabriqués (cf. Figure 11). Ce *framework* a été appliqué à la fois pour le système d'analyse et pour celui de simulation d'improvisation de Ramalho.

7.5. Harmonisation automatique

Le problème de l'harmonisation de chorals à quatre voix consiste à compléter une mélodie de départ donnée en écrivant les 3 autres voix de manière à respecter les règles d'harmonie et de contrepoint. Ces règles sont bien connues et expliquées longuement dans de nombreux traités d'harmonie, et ce, depuis le 17e siècle (voir, par exemple l'ouvrage de référence (Bitsch, 1957), ou la synthèse effectuée par Rémy Mouton dans sa thèse (Mouton, 1995), à l'usage des informaticiens non musiciens). C'est un phénomène unique dans l'histoire de l'Art. Le traité d'harmonie constitue même un véritable style littéraire (Corres, 1996).

Ce problème (en fait cette famille de problèmes, allant du chant donné à la basse chiffrée ou non chiffrée) est aussi un « grand classique » en informatique musicale et notamment comme illustration de la puissance des techniques de satisfaction de contraintes : Ebcioglu (Ebcioglu, 1993), Tsang (Tsang & Aitken, 1991), Ovans (Ovans, 1992), Ballesta (Ballesta, 1994). Nous avons nous aussi traité ce problème, mais cette fois en le considérant sous l'aspect « contraintes et objets », le prenant comme problème exemple typique.

Notre approche, avec Pierre Roy a consisté à suivre globalement le même cheminement qu'avec le système NéOpus. Partant des objets de base qui représentent les concepts de la musique tonale (la bibliothèque MusES, Cf. 7.3), nous avons construit un *framework* pour la représentation de contraintes - le système BackTalk, voir section 5 - et l'avons appliqué au problème de l'harmonisation automatique. Nous avons montré que « partir des objets » au lieu de « partir des contraintes » conduit à deux conceptions du problème radicalement différentes, avec un net avantage pour la première approche, dans le cas du problème de l'harmonisation (Pachet & Roy, 1995a; Pachet & Roy, 1995b). Ce résultat permet de proposer une conception objets très avantageuse dans tous les cas de problèmes de configuration, à trois niveaux, dans lesquels on doit poser des contraintes intra individu, inter individus, et entre groupes d'individus (voir section 5.2.2).

La Figure 12 montre un exemple de mélodie à harmoniser, et la Figure 13 une solution possible calculée par notre système, qui respecte les règles les plus fréquentes.

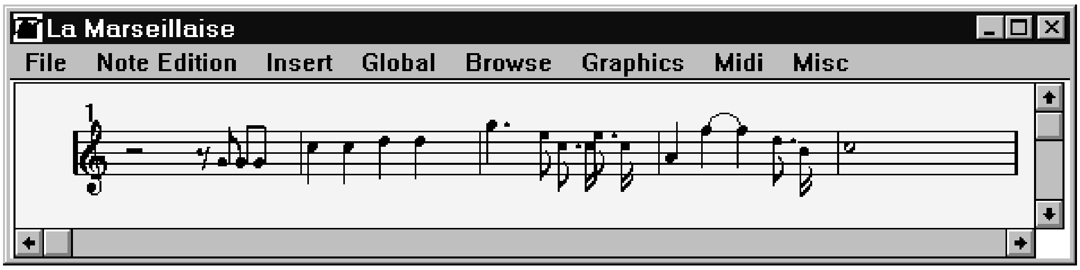


Figure 12. Une mélodie donnée.

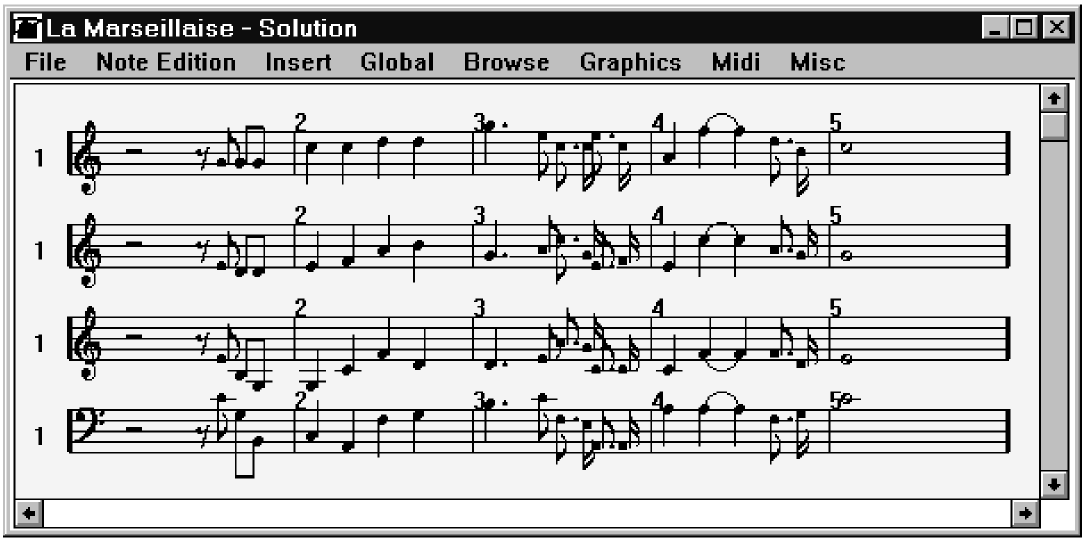


Figure 13. Une solution calculée par BackTalk et MusES pour la mélodie de la Figure 12.

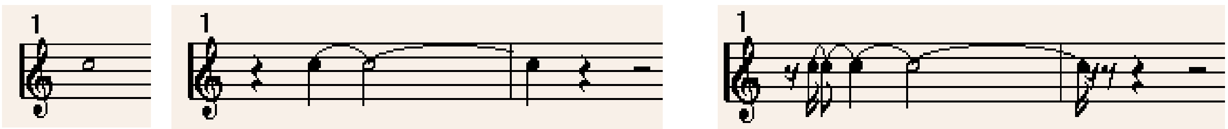
7.6. Editeurs graphiques

Le domaine des éditeurs graphiques est important dans l'histoire de la programmation par objets. Les technologies à objets ont provoqué une véritable révolution dans ce domaine, en simplifiant brutalement leur conception. Aujourd'hui encore, bon nombre des exemples utilisés pour illustrer les notions de *patterns* et de *frameworks* viennent des éditeurs graphiques. C'est un domaine très structuré, riche en métaphores, et complexifiable à souhait, idéal pour engendrer des intuitions, et poser des problèmes.

Je me suis intéressé pendant ma thèse au problème des éditeurs graphiques, d'abord dans le cadre de projet Sagesse du Ministère de l'Intérieur, puis dans divers projets ou l'édition de réseaux était centrale (réseaux Rete, réseaux de Petri, réseaux géographiques). J'ai à ce moment développé un *framework* pour fabriquer des éditeurs graphiques en Smalltalk, qui a donné lieu au système EdiTalk (Borne & Pachet, 1992) (Pachet, 1990a) utilisé au Laforia par plusieurs équipes.

La musique pose des problèmes particuliers en ce qui concerne l'édition graphique, auxquels il est difficile de rester insensible. Je me suis intéressé à la construction d'éditeurs de partitions interactifs véritablement utilisables, pour des raisons purement pratiques au début : il s'agissait de rentrer facilement de petites mélodies en MusES pour réaliser nos expérimentations (condition *sine qua none* de son utilisation effective), puis pour des raisons plus fondamentales, un éditeur graphique devant nécessairement posséder beaucoup de connaissances complexes sur la notation. Les problèmes posés sont de plusieurs nature : représentation graphiques complexes, contraintes fortes d'interdépendance des objets musicaux, efficacité. Mais le problème fondamental est qu'aucun des formalismes actuellement employés (y compris ceux que nous avons développés) ne permet de réduire cette complexité.

Pour donner une idée de la complexité des objets mis en jeu dans ces éditeurs, nous l'illustrons sur un exemple : 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 métrique. 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 14 (1) [11]. Lorsque cette même note commence sur le deuxième temps, on ne peut plus l'afficher comme telle, et ce 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 14 (2)). 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 14 (3)[12]. 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...



(1) (2) (3)

Figure 14. Trois représentations graphiques pour la « même » note.

De nombreuses recherches exploitent les formalismes de *propagation de contraintes* pour proposer des *frameworks* dans lesquels de tels éditeurs sont plus faciles à construire (Brant, 1995; Vlissides & Linton, 1990). L'application de ces techniques à des éditeurs de partition n'est pourtant pas immédiate. 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 s'accommodent mal 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. (Rader, 1996; Read, 1969). Enfin, les partitions réalistes comportent plusieurs centaines de notes, ce qui exclut en pratique l'utilisation de ces techniques.

Trouver une conception par objets d'un tel éditeur qui permette d'assurer un bon compromis entre interactivité et calculs automatiques reste donc un problème d'actualité. La solution choisie pour les éditeurs de MusES a consisté à définir trois niveaux d'objets pour représenter les notes. D'une part les *modèles* de notes, indépendants de toute représentation graphique, et fournis

---

[11] En fait il s'agit plus précisément d'une ellipse légèrement inclinée, et dont le trait est plus plein sur les bords que sur les sommets.

[12] Cette troisième représentation graphique est habituellement simplifiée, mais cette simplification complique en fait le problème !

directement par la bibliothèque MusES (instances de `OctaveDependentNote`). Pour les raisons évoquées plus haut, on ne peut établir de correspondance directe entre *un* modèle de note et *un* objet graphique. Pour chaque modèle de note on construit donc un objet intermédiaire non affiché, responsable de gérer la collection des notes graphiques effectivement affichées. Enfin, le troisième niveau est constitué des objets graphiques eux-mêmes. Cette indirection entre modèle et objet graphique permet de résoudre notre problème des notes fantômes, mais complexifie notablement l'architecture MVC.

7.7. Classification de sons

Un des sujets abordés lors de mon stage à l'IRCAM, était le problème de la programmation de synthétiseurs, en particulier du DX-7 de Yamaha, qui connaissait alors un succès commercial extraordinaire. J'ai eu la chance d'effectuer mon stage aux côtés de David Bristow, un des spécialistes de la programmation de patches en synthèse par modulation de fréquence. Son expertise était fascinante, et l'idée de la représenter en machine, pour la mettre à disposition des non-experts était très tentante.

J'ai eu l'occasion de revenir sur ce problème en 1994, avec Pierre-Yves Rolland, étudiant du DEA Atiam de l'Ircam à qui j'ai proposé de travailler sur ce sujet, sur un synthétiseur différent et plus simple (Korg 05/RW, la mode de la synthèse FM est aujourd'hui passée). Aidés par Jean-François Perrot, le résultat de ce travail fut de combiner les capacités de classification des logiques descriptives, le système *Back* (Hoppe et al., 1993), avec les capacités de représentation par objets de Smalltalk. L'idée de base était de classer les patches décrivant les sons du synthétiseur non pas en fonction de caractéristiques subjectives, mais en fonction des transformations qu'un expert sait leur appliquer.

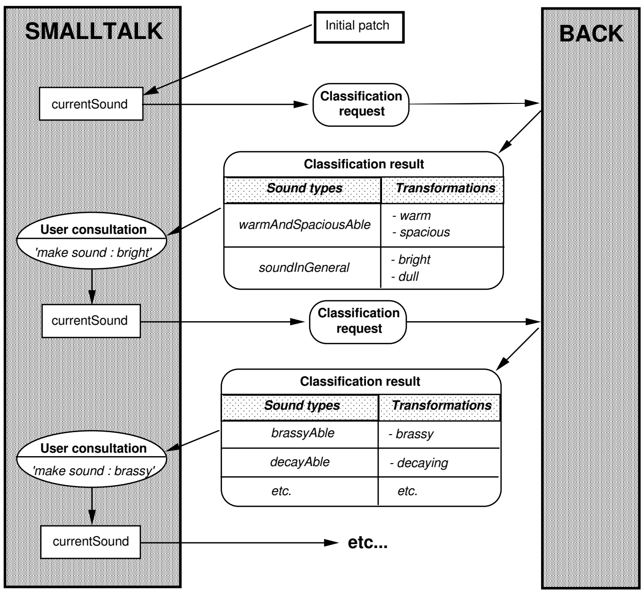


Figure 15. Le schéma bicéphale du système de classification de sons.

Un exemple typique de transformation experte est «pour rendre un son plus *gros*, on peut le doubler par une copie conforme, et désaccorder très légèrement sa copie ». Ces transformations définissent alors une taxonomie, adéquatement représentée dans le formalisme des logiques de description. Chacun des concepts de cette taxonomie représente une classe de sons à laquelle est associée, par définition même de la classe, une liste de transformations possibles. Par exemple, à la classe *Brassyable* est associée une liste de transformations permettant de rendre un son plus « cuivré ». Le succès de cette entreprise dépend alors de la capacité d'un expert à 1) décrire des classes de sons en fonction des paramètres du synthétiseur, et 2) décrire les transformations elles-mêmes, en fonction des actions primitives comprises par le synthétiseur. Pour le cas des synthétiseurs suffisamment simples comme celui choisi, ces deux postulats semblent vérifiés.

Du point de vue de la représentation des connaissances, le système repose sur une vision bicéphale du processus de raisonnement. La partie classificatoire est écrite en Back, et consiste à classer un son en fonction de ses caractéristiques dans la taxonomie pré-établie. Les transformations elles-mêmes sont représentées par des méthodes définies dans une classe Smalltalk (`Patch`). Ces méthodes définissent les transformations sous la forme d'actions de modifications de paramètres, directement comprises par le synthétiseur (messages dits *Midi system exclusive*).

Le cycle d'interaction est alors le suivant. Un son de départ est choisi par le programmeur, puis sa représentation est classifiée par Back. Le résultat est une classe de sons, à laquelle est associée une liste de noms de transformations possibles (plus chaud, plus brillant, etc.). Cette liste est alors présentée à l'utilisateur, qui en choisit une. Une fois la transformation choisie, Back passe la main à Smalltalk, qui exécute la *méthode* correspondant à la transformation, modifiant ainsi l'objet représentant le patch, et effectuant les actions correspondants sur le synthétiseur, par une interface Midi. Le son ainsi obtenu est alors renvoyé à Back pour un nouveau cycle, jusqu'à ce que le programmeur décide d'arrêter (cf. Figure 15).

Les premiers résultats furent encourageants (Rolland & Pachet, 1995a; Rolland & Pachet, 1995b; Rolland & Pachet, 1995c), et l'idée mérite d'être explorée plus avant, sur des synthétiseurs plus complexes (synthèse FM de deuxième génération, ou synthèse par modèles physiques). Par ailleurs, un tel système, pour être véritablement utilisable devrait être intégré à une interface plus intelligente, par exemple basée sur nos résultats en reconnaissance de plan (8.2.2).

8. Perspectives

Nous avons présenté ici nos travaux sous un angle particulier : l'intégration de mécanismes et techniques d'intelligence artificielle dans le cadre de la programmation par objets. Ces travaux ont pour la plupart comme résultat concret des propositions de réification d'objets abstraits : règles, règles déclenchables, évaluateurs, conseils, tâches, notes, objets temporels, intervalles, modèles et méta-modèles, classes et méta-classes, contraintes, heuristiques. Chacun de ces objets échappe au cadre des langages à objets et nous avons recherché des moyens de les modéliser pour mieux comprendre leur structure profonde, et proposer des moyens de les représenter.

8.1. Des idées à explorer

Mes projets à moyen terme consistent à continuer de développer les idées présentées ici, en leur donnant plus d'épaisseur. J'indique ici trois directions de recherche que j'aimerais développer avec l'aide de mes étudiants. Une direction exploratoire : étude du rapport entre métamodélisation et métaclasses; expérimentale : étude de nouvelles métaphores pour les éditeurs graphiques, et théorique : élaboration de modèles formels rendant compte d'un certain nombre d'expérimentations déjà effectuées.

1. *Modélisation et instanciation*

Un des axes non encore explorés est celui du recours à la programmation par métaclasse pour ce type d'approche. Une implémentation de *Metagen* en ClassTalk devrait se révéler ici extrêmement intéressante. Un de mes projets consiste donc à explorer la version de ClassTalk "moderne" pour étudier la méta-modélisation au sens de Blain. La question posée est alors : en quoi un méta modèle est-il différent d'une métaclasse ? L'élément de réponse apporté lors du workshop OOPSLA (Mili et al., 1995) fut que la méta-modélisation et l'instanciation sont deux directions très proches, mais qui proposent chacune un "biais" différent sur la représentation d'un modèle. Le biais s'écarte ainsi de plus en plus au fur et à mesure que l'on remonte cette relation (cf. Figure 16). La question reste ouverte et, ici encore, seules des expérimentations substantielles pourront nous donner des éléments de réponse.

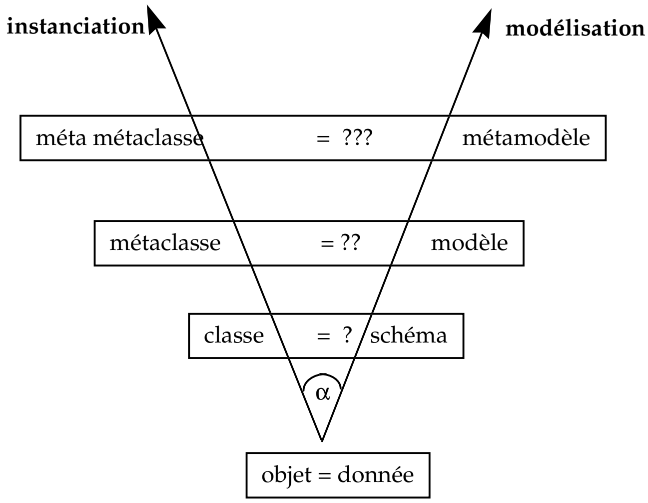


Figure 16. Instanciation et modélisation. On peut confondre donnée et objet, mais quand on remonte aux origines, la confusion devient de plus en plus gênante : quelle est la valeur de l'angle α ?

2. *Editeurs graphiques et Programmation visuelle*

Je suis convaincu que la notation musicale est un domaine riche en métaphores puissantes applicables à de nombreux systèmes à bases de connaissances, en particulier temporelles. Le système de notation de la musique occidentale traditionnelle est le résultat d'une longue évolution. Bien que périodiquement remis en question, ce système de notation est extrêmement astucieux.

Une application directe de la métaphore de la partition est celle proposée par (Domingue & Eisenstadt, 1991) pour la visualisation des déclenchements de bases de règles en chaînage avant. L'idée consiste à visualiser les relations *causales* entre les règles, au lieu de produire des traces des séquences de déclenchements, qui sont en général inexploitables. Cette interface permet en particulier de visualiser les règles déclenchables non déclenchées, information souvent cruciale pour comprendre l'exécution d'une base de règles. Cette métaphore à été reprise pour la version de NéOpus développée par la société OTI. L'outil sert alors de base pour spécifier des points d'arrêt de manière graphique. On peut alors aisément spécifier des conditions portant sur les règles déclenchables ou déclenchées, comme par exemple : s'arrêter au 8ème déclenchement de la règle r4, ou bien dès que les règles r1 et r2 sont simultanément déclenchables. Par ailleurs, cet outil permet aussi de s'affranchir de la contrainte "tout ou rien" en mode "pas à pas" : on peut décider de se mettre en mode pas à pas dès qu'une certaine règle devient déclenchable, et jusqu'à ce qu'une autre le soit, ou toute autre condition similaire.

Ces résultats montrent la puissance de cette notation. J'aimerais continuer des recherches en ce sens, notamment pour les systèmes de satisfaction de contraintes, qui posent des problèmes de visualisation de traces particulièrement ardus.

De manière plus générale, j'aimerais explorer les possibilités de la programmation visuelle, notamment dans le domaine de la satisfaction de contraintes. La propagation de contraintes a traditionnellement été appliquée à l'édition graphique, mais les systèmes de satisfaction de contraintes sont restés terriblement textuels. Un des projets en cours étudie la manière dont on peut spécifier graphiquement des problèmes combinatoires, et générer automatiquement des spécifications de problèmes (BackTalk), qui sont ensuite exécutées de manière standard. De telles interfaces bien construites permettraient de rendre ces techniques plus facilement disponibles par des non-experts, et donc d'élargir leur champ d'application.

3. *Formalisation*

La formalisation en intelligence artificielle a souvent mauvaise presse. Formaliser revient souvent à faire disparaître les difficultés véritables, pour se concentrer sur celles que l'on sait résoudre. Il y a en effet des situations dans lesquelles il ne faut surtout pas formaliser. Les travaux récents sur les *patterns* en programmation (Gamma et al., 1994) en sont un exemple typique. Cependant, bien qu'elle soit souvent déplacée, voire dangereuse, la formalisation d'un certain nombre de travaux est parfois nécessaire. Nous identifions trois points sur lesquels nous avons travaillé de manière expérimentale et qui mériteraient maintenant un traitement formel.

1. La reconnaissance de plans et les systèmes conseillers. Les premiers prototypes et applications montrent qu'il est possible de formaliser la notion de conseil, comme entité du deuxième ordre calculée à partir de l'analyse d'une séquence d'action (le lien avec les grammaires attribuées paraît ici naturel).

2. Les *patterns* de règles dans les systèmes d'analyse par règles et objets.

3. Les propriétés des intervalles de temps circulaires, dans la suite des travaux effectués avec Jean Carrive (Pachet & Carrive, 1996). Ce travail en cours tente de rapprocher nos premiers résultats avec le formalisme général des réseaux de contraintes temporels.

8.2. Des contextes nouveaux

Cependant, la nature des expérimentations requises, la taille des programmes à réaliser, et la nécessité de procéder à des validations dans plusieurs domaines d'application entraînent la fin d'une conception de la recherche mono-chercheur, et centrée sur des expérimentations dans des "micro-mondes". La recherche en représentation de connaissances requiert aujourd'hui la mise en place de projets et de collaborations qui doivent dépasser le stade du chercheur individuel, voire de l'équipe. Un de mes objectifs est donc de mettre en place des collaborations, en particulier européennes, permettant de développer et valider de nouvelles idées sur une autre échelle. J'en identifie ici deux, dans la continuité de mes travaux.

1. *Conseillers distribués et collaboratifs*

L'enseignement à distance est un des enjeux majeurs des années à venir. La télé-université à Montréal développe depuis plusieurs années des activités de recherche dans ce domaine et a acquis une renommée internationale, grâce à plusieurs projets dont le "Campus virtuel". Une des applications prometteuse de l'architecture EpiTalk est de fabriquer des conseillers distribués, produisant des conseils sur des activités collaboratives. Un projet en cours (avec des étudiants du DESS GLA) consiste à réaliser une plate-forme permettant l'espionnage à distance, en utilisant la plate-forme RpcTalk (Wolinski, 1994). La première application de cette plate-forme est un système conseiller sur l'organisation de colloque.

Par ailleurs je participe à un projet expérimental d'enseignement à distance avec la Télé-Université entre Montréal, Paris et La Réunion. Un même cours, sur support Web, est donné à trois classes physiquement éloignées. Les activités de tutorat, conseil et contrôle sont réalisées indifféremment par des tuteurs humains des trois zones, selon une architecture propre au LICEF.

2. *Retour à l'objet musical*

Je cherche à approfondir les travaux en représentation musicale et à les appliquer à des problèmes soulevés par les techniques et besoins modernes en interaction homme-machine.

Cette problématique m'a amené à fréquenter la communauté de l'Informatique Musicale française, qui a pris depuis quelques années un statut nouveau, comme en témoigne le succès des Journées d'Informatique Musicale (JIM) tenues régulièrement depuis 1994, et dont j'ai organisé la deuxième édition (Pachet et al., 1995a). Ces conférences connaissent aujourd'hui un vif succès, la quatrième édition aura lieu à Lyon, et une synthèse des trois premières sous forme de livre est en cours (Assayag et al., 1997).

Le projet EpiTalk montre qu'il est aujourd'hui possible de mettre à la portée d'utilisateurs non informaticiens des programmes ayant une capacité d'analyse non triviale. Cette faculté d'analyse permet d'améliorer grandement les interfaces, au sens large du terme. Ces techniques ces résultats dépassent en outre largement le contexte des systèmes éducatifs : dès lors que l'on peut exhiber des arbres de tâches raisonnables, l'architecture est applicable et permet de construire rapidement des systèmes capables de produire des conseils de manière systématique, tout en laissant la possibilité d'insérer des conseils spécifiques. Une des applications de ces techniques que j'aimerais poursuivre est l'interface de synthétiseurs : les techniques de reconnaissance de plan permettraient en effet, combinées avec celles que nous avons développées pour la classification de sons (Cf. 7.7), d'aboutir enfin à des interfaces intelligentes et adaptables.

D'autre part, je participe au projet "Partition Intérieure" avec Francis Rousseaux, Jacques Siron et d'autres partenaires. Ce projet a comme objectif de participer à l'effort de recherche pour la fabrication de CD-ROM interactifs à forte valeur ajoutée. Nous proposons un projet pluridisciplinaire articulé autour de l'ouvrage La Partition Intérieure de Jacques Siron (Siron, 1992). Ce livre - édité en France depuis 1992, et qui rencontre un succès éditorial considérable - aborde la question de l'improvisation par le biais d'une démarche pédagogique non-linéaire et multiforme, qui favorise les chemins de traverse et les évocations musicales aussi bien sonores que graphiques, gestuelles ou poétiques. Notre projet a pour but de concevoir, réaliser et diffuser une refonte multimédia et multimodale de La Partition Intérieure (Rousseaux & Pachet 1997a, 1997b, 1997c), (Rousseaux et al. 1997).

Enfin, je codirige une thèse CIFRE avec l'INA (Institut National d'Audiovisuel), sur le thème de la visualisation de la bande son dans les logiciels de dessins animés (Jean Carrive). Le problème abordé est celui de la manipulation du son en rapport avec l'image, et plus précisément l'étude de la représentation informatique (par objets) des *objets sonores* relatifs au dessin animés. Dénommés "acteurs son", ces objets complexes doivent obéir à des contraintes précises, dictées par les besoins des concepteurs de *storyboard*. D'une certaine manière, il s'agit ici de trouver de bonnes représentations, informatiques et graphiques, à l'*objet musical* de Pierre Schaeffer.

Ici encore, la musique sert de terrain d'investigation privilégié, mais l'objectif final est toujours le même : exploiter pleinement les ressources de l'informatique, en particulier de l'intelligence artificielle et génie logiciel, pour aider les utilisateurs à réaliser des tâches de plus en plus complexes, et de moins en moins formalisées.

9. Table des Figures

-

10. Bibliographie

10.1. Bibliographie personnelle

Revues, chapitres de livre

Assayag, G., Chemillier, M., Pachet, F., and Serra, M.-H. (1997). *Journées d'Informatique Musicale, le livre*, Hermes.

Borne, I., and Pachet, F. (1992). "From Object-Oriented Design to Visual Programming." *European Journal of Engineering and Education*, 17(2), 195-201.

Dojat, M., and Pachet, F. (1992a). "NéoGanesh, un système extensible à base de connaissances pour la ventilation artificielle des patients." *Innovation et Technologie en Biologie et Medecine*, 14, 267-279.

Dojat, M., and Pachet, F. (1995a). "Effective Domain-Dependent Reuse in Medical Knowledge Bases." *Computers and Biomedical Research*(28), 403-432.

Dojat, M., Pachet, F., Guessoum, Z., Touchard, D., Harf, A., and Brochard, L. (1997). "NéoGanesh: a Working System for the Automated Control of Assisted Ventilation in ICUs." *Artificial Intelligence in Medicine. Special issue on Decision Support in the Operative Theatre and Intensive Care.*

Giroux, S., Pachet, F., Paquette, G., and Girard, J. (1995). "Des systèmes conseillers épiphytes." *Revue d'intelligence artificielle*, 9(2), 165-190.

Mili, H., and Pachet, F. (1995a). "Régularité, génération de documents et Cyc." *Revue d'intelligence artificielle*, 9(2), 139-164.

Mili, H., and Pachet, F. (1995b). "Regularity, Document Generation and Cyc." HyperText and HyperMedia, R. Rada and K. Tochtermann, eds., World Wide Publishing, 171-206.

Pachet, F. (1995b). "On the embeddability of production rules in object-oriented languages." *Journal of Object-Oriented Programming*, 8(4), 19-24.

Pachet, F. (1997a). "Computer Analysis of Jazz Chord Sequences. Is Solar a Blues ?" *Contemporary Music Review*, to appear.

Pachet, F. (1997b). "Musique et objets." Langages à Objets, J. Euzenat and A. Napoli, eds., Hermès, Paris.

Pachet, F., Djamen, J.-Y., Frasson, C., and Kaltenbach, M. (1996a). "Production de conseils pertinents exploitant les relations de précédence et de composition dans un arbre de tâches." *Sciences et Techniques Educatives*, 3(1), 43-75.

Pachet, F., Ramalho, G., and Carrive, J. (1996b). "Representing temporal musical objects and reasoning in the MusES system." *Journal of New Music Research*, 25(3), 252-275.

Paquette, G., Pachet, F., and Giroux, S. (1994). "EpiTalk, un outil générique pour la construction de systèmes conseillers." *Sciences et Techniques Educatives*, 1(3), 305-336.

Paquette, G., Pachet, F., Giroux, S., and Girard, J. (1996). "EpiTalk, Generating Advisor Agents for Existing Information Systems." *Artificial Intelligence in Education*, 7(3-4), 149-379.

Rolland, P.-Y., and Pachet, F. (1995a). "A framework for representing knowledge about synthesizer programming." *Computer Music Journal*, 20(3), 47-58.

Roy, P., and Pachet, F. (1997c). "Reifying Constraint Satisfaction in Smalltalk." *Journal of Object-Oriented Programming*, July/August 1997, vol. 10, n. 4, pp. 43-51.

Conférences

Dojat, M., and Pachet, F. (1992b). "NéoGanesh: an Extendable Knowledge-Based System for the Control of Mechanical Ventilation." *14th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Paris, 920-921.

Dojat, M., and Pachet, F. (1992c). "Representation of a Medical Expertise Using the Smalltalk Environment: Putting a Prototype to Work." TOOLS Europe '7, G. Heeg, B. Magnusson, and B. Meyer, eds., Prentice-Hall, Dortmund (Germany), 379-389.

Dojat, M., and Pachet, F. (1995b). "Representing Medical Context Using Rule-Based Object-Oriented Programming Techniques." *Artificial Intelligence in Medicine Europe*, Pavia (It), 423-424.

Djamen, J.-Y., and Pachet, F. (1997). "Advice with Part-Whole and Precedence Relations in Task Graphs for Intelligent Tutoring Systems." *FLAIRS'97 Special track on Intelligent Tutorial Systems*.

Giroux, S., Pachet, F., and Paquette, G. (1994b). "Des systèmes multi-agents épiphytes." *Deuxièmes Journées Francophones "intelligence artificielle Distribuée & Systèmes Multi-Agents"*, Grenoble, 211-222.

Giroux, S., Pachet, F., and Senteni, A. (1993). "Contrôle descendant versus structuration ascendante." *61 ème congrès de l'ACFAS*, Rimouski, Canada, 408.

Giroux, S., Paquette, G., Pachet, F., and Girard, J. (1996). "EpiTalk, a Platform for Epiphyte Advisor Systems Dedicated to Both Individual and Collaborative Learning." *ITS-96*, Montréal.

Karsenty, S., and Pachet, F. (1995). "Un mécanisme hiérarchique de répétition et prédiction de tâches." IHM 95, Toulouse, 169-175.

Pachet, F. (1989). "A Practical Use of Metaclasses." TOOLS' 89, Paris, 233-242.

Pachet, F. (1990b). "Mixing Rules and Objects: an Experiment in the World of Euclidean Geometry." ISCIS V, Nevsehir, Turquie, 797 - 805.

Pachet, F. (1991b). "A meta-level architecture for analysing jazz chord sequences." *International Conference on Computer Music 1991*, Montréal, 266-269.

Pachet, F. (1991e). "Reasoning with objects : the NéOpus environment." *Conférence East EurOOpe*, Bratislava, Tchécoslovaquie, 72-87.

Pachet, F. (1991f). "Representing Knowledge Used by Jazz Musicians." *International Conference on Computer Music 1991*, Montréal, 285-288.

Pachet, F. (1992b). "Rule Base Inheritance." *Colloque "Représentations Par Objets"*, La grande Motte, 187-200.

Pachet, F. (1994a). "The MusES system : an environment for experimenting with knowledge representation techniques in tonal harmony." *First Brazilian Symposium on Computer Music - SBC&M '94*, Caxambu, Minas Gerais, Brazil, 195-201.

Pachet, F. (1994b). "An object-oriented representation of pitch-classes, intervals, scales and chords." *Premières Journées d'Informatique Musicale*, LaBRi, Université de Bordeaux, 19-34.

Pachet, F. (1994f). "Vers un modèle du raisonnement dans les langages à objets." *Colloque Langages et Modèles à Objets*, Grenoble, 111-123.

Pachet, F. (1996). "Objets, règles et patterns." *GDR Programmation, pôle "Objets"*, Orléans.

Pachet, F., and Carrive, J. (1996). "Propriétés des intervalles temporels circulaires et application à l'analyse harmonique automatique." *Troisièmes Journées d'Informatique Musicale, JIM'96*, Ile de Tatihou, 230-247.

Pachet, F., Giroux, S., and Paquette, G. (1994). "Pluggable Advisors as Epiphyte Systems." *Calisce '94 (Computer Aided Learning in Science and Engineering)*, Paris, 167-174.

Pachet, F., and Mili, H. (1994). "Exploiting regularity in Cyc for text generation." *World Congress on Expert Systems II*, Lisbonne, 711-719.
Pachet, F., and Perrot, J.-F. (1994b). "Rule Firing with Metarules." *Software Engineering and Knowledge Engineering - SEKE '94*, Jurmala, Latvia, 322-329.
Pachet, F., Ramalho, G., Carrive, J., and Cornic, G. (1995b). "Representing temporal musical objects and reasoning in the MusES system." *International Congress in Music and Artificial Intelligence*, Edinburgh, 33-48.
Pachet, F., and Roy, P. (1995a). "Integrating constraint satisfaction techniques with complex object structures." *15th Annual Conference of the British Computer Society Specialist Group on Expert Systems, ES'95*, Cambridge, 11-22.
Pachet, F., and Roy, P. (1995b). "Mixing constraints and objects: a case study in automatic harmonization." TOOLS Europe '95, I. Graham, B. Magnusson, and J. M. Nerson, eds., Prentice-Hall, Versailles, 119-126.
Pachet, F., Wolinski, F., and Giroux, S. (1995c). "Spying as an object-oriented programming paradigm." TOOLS Europe '95, I. Graham, B. Magnusson, and J. M. Nerson, eds., Prentice-Hall, Versailles, 109-118.
Ramalho, G., and Pachet, F. (1994). "From Real Book to Real Jazz Performance." *International Conference on Music Perception and Cognition (ICMPC)*, Lièges (Belgium), 349-350.
Rolland, P.-Y., and Pachet, F. (1995c). "Représentation de connaissances et programmation de synthétiseurs." *Deuxièmes Journées d'Informatique Musicale*, Paris, 123-132.
Rousseaux, F., and Pachet, F. (1997). "L'espace dans le projet « La partition intérieure interactive »." *Colloque Musique/Philosophie*, Université Paris IV, Paris.
Rousseaux, F., and Pachet, F. (1997b). "Acquisition des connaissances et improvisation: la Partition Intérieure Interactive." Journées d'Informatique Musicale, JIM 97, Lyon.
Rousseaux, F., and Pachet, F. (1997c). "Distinction and characterization of subjective vs objective aspects for the design of artificial assistance systems in "AI and Music" field." IJCAI 97 Workshop on Music and AI, Nagoya.
Rousseaux, F., Meunier, J.-G., and Pachet, F. (1997). "Comment penser un système d'assistance artificiel quand l'improvisation est au coeur de l'activité à modéliser ? Le cas du projet "La Partition Intérieure Interactive"." IC '97 (Ingénierie des Connaissances), Roscoff.
Roy, P., and Pachet, F. (1996a). "Contraintes et mots croisés." *GDR Programmation*, Orléans.
Roy, P., and Pachet, F. (1997b). "Conception de problèmes par objets et contraintes." *Journées francophones des langages applicatifs*, Lyon, 169-187.

Workshops de conférence

Dojat, M., and Pachet, F. (1995c). "Three Compatible Mechanisms for Representing Medical Context Implicitly." *IJCAI Workshop on "Context in Knowledge and Reasoning Modelling"*, Montréal.
Giroux, S., Pachet, F., and Desbiens, J. (1994a). "Debugging multi-agent systems: a distributed approach to events collection and analysis." *Canadian Workshop on Distributed Artificial Intelligence - CWDAI '94.*, Banff, Alberta, Canada.
Mili, H., Pachet, F., Benhyaya, I., and Eddy, F. (1995). "Report on the OOPSLA'95 Workshop on Metamodeling." *Addendum to the OOPSLA'95 proceedings*, Austin, Texas, 87-91.
Mouton, R., and Pachet, F. (1995). "Numeric vs symbolic controversy in automatic analysis of tonal music." *IJCAI'95 Workshop on Artificial Intelligence and Music*, Montréal, 32-40.
Pachet, F. (1987a). "MusES - a Musical Expert System using Experkit in an Object-Oriented Environment." *Expert system workshop and seminar*, Kuala-Lumpur (Malaysia).

Pachet, F., and Giroux, S. (1995). “Building plan recognition systems on arbitrary applications: the spying technique.” *IJCAI'95 Workshop on "Next generation of plan recognition systems"*, Montréal, Canada.
Pachet, F., and Perrot, J.-F. (1994a). “Report on the NéOpus system.” *Workshop OOPSLA'94 on Embedded Object-Oriented Production Systems*, Portand, Oregon, 33-40.
Rolland, P.-Y., and Pachet, F. (1995b). “Modeling and Applying the Knowledge of Synthesizer Patch Programmers.” *IJCAI'95 Workshop on Artificial Intelligence and Music*, Montréal.

Divers (rapports internes, tutoriels)

Pachet, F. (1987b). “Vers un système expert de suivi d'improvisation.” , Laforia-IBP/IRCAM.
Pachet, F. (1990a). “EdiTalk, mode d'emploi.” , ACKIA, rapport interne.
Pachet, F. (1991a). “Du bon usage des méta-règles en NéOpus.” *91/16*, Laforia-IBP.
Pachet, F. (1991c). “NéOpus mode d'emploi.” *91/14*, Laforia-IBP, Paris.
Pachet, F. (1991d). “Pour en finir avec le singe et les bananes.” *91/15*, Laforia-IBP.
Pachet, F. (1992a). “Représentation de connaissances par objets et règles : le système NéOpus,” Ph. D., Université Paris 6.
Pachet, F. (1994c). “Proceedings of the OOPSLA'94 Workshop on EOOPS.” *94/24*, Laforia-IBP.
Pachet, F. (1994d). “Report on the OOPSLA'94 Workshop on EOOPS.” *Addendum to the OOPSLA'94 proceedings*, Portland, Oregon, 87-91.
Pachet, F. (1994e). “Tutorial on "Epiphyte Programming in Smalltalk-80".” , Summer School of the European Smalltalk User Group, Cork, Ireland.
Pachet, F. (1995a). “NéOpus Users' Guide.” *95/23*, Laforia-IBP.
Pachet, F., and Assayag, G. (1995). “Dossier IA et musique.” *Bulletin de l'AFIA*, 23.
Pachet, F., Brown, F., and Mouton, R. (1995a). “Deuxièmes Journées d'Informatique Musicale.” *95/13*, Laforia-IBP.
Pachet, F., and Dojat, M. (1995). “Un framework pour la représentation de connaissances temporelles en NéOpus.” *95/19*, Laforia-IBP.
Paquette, G., Giroux, S., and Pachet, F. (1993). “Méthodes et outils de développement de systèmes conseillers dans les environnements de formation.” , LICEF, Télé-Université.
Roy, P., and Pachet, F. (1995). “Tutorial on Integration of Constraint Satisfaction in Object Oriented Languages.” , Cambridge, UK.
Roy, P., and Pachet, F. (1996b). “Tutorial on Integration of Constraint Satisfaction in Smalltalk.” , European Smalltalk User Group, Genève (Suisse).
Roy, P., and Pachet, F. (1997a). “BackTalk mode d'emploi.” *97/XX*, Laforia-IBP.
Roy, P., and Pachet, F. (1997d). “Tutorial on Constraint Satisfaction Problems and Objects.” , IJCAI 97, Nagoya.

10.2. Bibliographie du domaine

Alexe, C., and Gecsei, J. (1996). “A Learning Environment for the Surgical Intensive Care Unit.” *ITS*, Montréal, 439-447.

Allen, J. F. (1984). “Towards a general theory of action and time.” *Artificial Intelligence*, 23(2), 123-154.

Alvarez, I. (1992). “Explication morphologique: un mode d'explication fondé sur la géométrie,” Ph.D., Université Pierre & Marie Curie.

Amarel, S. (1966). “Comments on the Mechanization of Creative Processes.” *IEEE Spectrum*.

Atkinson, R., and Laursen, J. (1987). “Opus: A Smalltalk Production System.” *OOPSLA'87*, 377-387.

Ballesta, P. (1994). “Contraintes et objets: clefs de voûte d'un outil d'aide à la composition,” Ph. D., Université du Maine, Le Mans.

Balzer, R. (1990). “Looking for AI in software Engineering, an Application Perspective.” *Panel "AI and Software Engineering : Will the Twain Ever Meet ? in AAAI'90*, 1123-1129.

Barry, B., and McAffer, J. (1992). “ENVY/Expert : a rule-based programming system for Smalltalk.” *Canadian AI Conference*.

Benhouhou, A. (1993). “Un système de formation basé sur un simulateur et un générateur de scénarios.” *ICO '93*, Montréal, 29-36.

Benhouhou, A. (1996). “Une base de connaissances événementielle pour le guidage interactif des situations de crise,” Ph.D., Université Pierre & Marie Curie, Paris.

Benyahia, I. (1994). “Automatisation de la circulation des informations en temps réel,” Ph.D., Université Pierre & Marie Curie.

Bitsch, M. (1957). *Précis d'harmonie tonale*, Editions musicales Alphonse Leduc, Paris.

Blain, G., Revault, N., Sahraoui, H., and Perrot, J.-F. (1994). “A Metamodelization Technique.” *OOPSLA 94 Workshop on AI and Software Engineering*, Portland.

Borning, A. (1981). “The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory.” *ACM transaction on Programming Languages and Systems*, 3(4), 353-387.

Borning, A., and Freeman-Benson, B. (1995). “The OTI Constraint Solver: A Constraint Library for Constructing Interactive Graphical User Interfaces.” *CP'95*, Cassis, 624-628.

Borning, A., Freeman-Benson, B., and Wilson, M. (1992). “Constraint Hierarchies.” *Lisp and Symbolic Computation*, 2, 223-270.

Bouaud, J., and Voyer, R. (1996). “Langages à objets et langages de règles: étude critique et propositions d'intégration.” *Technique et Science Informatiques*, 15(6), 831-860.

Bourgeois, R. (1990). “ICEO : Intension, coréférences et objets dans la fédération de formalismes de spécification,” Ph.D., Paris 6, Paris.

Bourgine, P. (1989). “Le langage Promat et sa machine virtuelle.” *RFIA*, Paris.

Boynton, L., Lavoie, P., Y., O., Rueda, C., and Wessel, D. (1986). “MIDI-LISP, a Lisp-based music programming environment for the macintosh.” *International Computer Music Conference*, San Francisco, 183-186.

Brant, J. M. (1995). “HotDraw,” Master thesis, University of Illinois at Urbana-Champaign, Urbana-Champaign.

Briot, J. P. (1989a). “Actalk: a testbed for classifying and designing actor languages in the Smalltalk-80 environment.” *ECOOP'89*, Edinburgh (UK), 109-130.

Briot, J. P. (1989b). “Des objets aux acteurs. 1982-1989: 7 ans de réflexion. Habilitation à diriger des recherches. Mémoire de synthèse,” Habilitation, Paris 6, Paris.

Briot, J. P., and Guerraoui, R. (1996). “Objets pour la programmation parallèle et répartie: intérêts, évolutions et tendances.” *Techniques et Sciences Informatiques*, 15(6), 765-800.

Briot, J.-P., and Cointe, P. (1989). "Programming with Explicit Metaclasses in Smalltalk-80." *OOPSLA*, New Orleans, 419-432.
Cantwell Smith, B. (1991). "The owl and the electric encyclopedia." *Artificial Intelligence*, 47, 251-288.
Carré, B., Ducournau, R., Euzenat, J., Napoli, A., and Rechenmann, F. (1995). "Classification et objets: programmation ou représentation ?" *5e Journées Nationales du PRC-GDR intelligence artificielle*, 213-237.
Caseau, Y. (1994). "Constraint Satisfaction with an Object-Oriented Knowledge Representation Language." *Journal of Applied Intelligence*, 4, 157-184.
Chang, S. K. (1971). "The reconstruction of binary patterns from their projections." *Communications of the ACM*, 14, 21-25.
Charbonnel, S. (1990). "Etude et réalisations de l'environnement du générateur de systèmes experts NéOpus," Masters, CEMAGREF - Université de Nantes.
Codognet, P. (1995). "Programmation logique avec contraintes: une introduction." *Technique et Sciences Informatiques*, 14(6).
Codognet, P., and Diaz, D. (1996). "Compiling Constraints in clp(FP)." *Journal of Logic Programming*, 27(3), 195-226.
Cointe, P. (1984). "Implémentation et interprétation des langages orientés objets : application aux langages Smalltalk, ObjVlisp et Formes," Ph.D., Paris 6, Paris.
Cointe, P. (1987). "Metaclass are first class: the ObjVlisp model." *OOPSLA '87*, Orlando, 156-168.
Corres, C. (1996). *Ecritures de la musique*, PUF, Paris.
CycBookReviews. (1993). "Book reviews and response from Lenat & Guha." *Artificial Intelligence*, 61(1), 37-181.
Danforth, S., and Forman, I. R. (1994). "Reflections on Metaclass Programming in SOM." *OOPSLA*, Portland, 440-452.
Dechter, R., Meiri, I., and Judea, P. (1991). "Temporal constraints networks." *Artificial Intelligence*, 4, 61-95.
Djamen, J.-Y. (1995). "Architecture de Système Tutoriel Intelligent pour l'Analyse du Raisonnement de l'Apprenant," Ph.D., Université de Montréal, Montréal.
Dojat, M. (1994). "Contribution à la représentation d'expertises dynamiques médicales. Application en réanimation," Ph.D., Université de technologie de Compiègne.
Dojat, M., Harf, A., Touchard, D., Laforest, M., Lemaire, F., and Brochard, L. (1996). "Evaluation of a knowledge-based system providing ventilatory management and decision for extubation." *Am. J. Respir. Crit. Care Med*, 153, 997-1004.
Dojat, M., and Sayettat, C. (1996). "A realistic model for temporal reasoning in real-time patient monitoring." *Applied Artificial Intelligence*, 10(2), 121-143.
Domingue, J., and Eisenstadt, M. (1991). "A new metaphor for the graphical explanation of forward-chaining rule execution." *IJCAI 91*, Sydney (Australia), 129-134.
Ducournau, R. (1996). "Sémantique de la représentation par objets: les systèmes classificatoires." Objets, tendances et évolutions., INRIA.
Dufresne, A., Gecsei, J., Cromp, P., and Alexe, C. (1995). "Cardexp: a graphical and hypermedia interface to a knowledge base to learn decision making." *InterAct'95*.
Ebcioglu, K. (1993). "An Expert System for Harmonizing Four-Part Chorales." Machine Models of Music, S. M. Schwanauer and D. A. Levitt, eds., MIT Press, 385-401.
Euzenat, J., and Rechenmann, F. (1995). "Shirka, 10 ans, c'est Tropes ?" *LMO*, Nancy, 13-34.
Ferber, J. (1987). "L'Experkit, générateur de systèmes experts." , ACT Informatique, Paris.
Ferber, J. (1989). "Objets et agents: une étude des structures de représentation et de communications en intelligence artificielle," Thèse d'Etat, Paris 6, Paris.

Fillod, O. (1995). “Un système qui apprend à résoudre des problèmes combinatoires en utilisant des métaconnaissances,” Master, Université Pierre & Marie Curie.

Forgy, C. L. (1982). “Rete: A fast algorithm for the many pattern/ many object pattern match problem.” *AI*, 19, 17-37.

Forte, A.-M. (1991). “Système basé sur la connaissance pour l'identification, la caractérisation et la mise en correspondance d'entités anatomiques et fonctionnelles en imagerie médicale multimodalité,” Ph.D., Université François Rabelais, Tours.

Forte, A.-M., Bernardet, M., Lavaire, F., and Bizais, Y. (1992). “Object-Oriented versus Logical Conventional Implementation of a MMIIS.” *SPIE'92, Medical Imaging VI*, Newport Beach, Ca.

Freeman-Benson, B. (1990). “Kaleidoscope: Mixing Objects, Constraints, and Imperative Programming.” *ECOOP/OOPSLA 90*, Ottawa (Ca), 77-88.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison-Wesley.

Gecsei, I., and Frasson, C. (1994). “SAFARI: an Environment for Creating Tutoring Systems in Industrial Training.” *Ed Media '94*, Vancouver, 15-20.

Gensel, J. (1995). “Integrating Constraints in an Object-Based Knowledge Representation System.” Lecture Notes in Computer Science n. 923, Springer-Verlag, 67-83.

Giroux, S. (1993). “Agents et systèmes: une nécessaire unité,” Ph.D., Université de Montréal, Montréal.

Graubé, N. (1989). “Metaclass compatibility.” *OOPSLA*, New Orleans, 305-316.

Greussay, P. (1973). “Modèles de descriptions symboliques en analyse musicale,” Ph.D., Université Paris 8.

Greussay, P. (1985). “Exposition ou exploration: graphes Beethovéniens.” Quoi ?, Quand ?, Comment ? La recherche musicale, Christian Bourgois, IRCAM, Paris, 165-183.

Guessoum, Z. (1996). “Un environnement opérationnel de conception et de réalisation de systèmes multi-agents,” Ph.D., Université Pierre & Marie Curie, Paris.

Guha, R. V., and Lenat, D. B. (1994). “Enabling Agents to Work Together.” *Communications of the ACM*, 37(7), 127-142.

Hayes-Roth, B. (1985). “A blackboard Architecture for Control.” *Artificial Intelligence*, 26, 251-321.

Hofstadter, D. (1985). *Gödel, Escher, Bach*, InterEditions, Paris.

Hofstadter, D. (1995). *Fluid Concepts and Creative Analogies*, Basic Books.

Holland, S. (1989). “Artificial intelligence, Education and Music,” Ph.D., Open University, Milton Keynes (GB).

Hoppe, T. C., Kindermann, J., Quantz, A., Schmiedel, M., and Fischer, R. (1993). “Back V5 Tutorial & Manual.” , Institut für Software und Theoretische Informatik, Berlin, Germany.

Horowitz, D. (1995). “Representing Musical Knowledge: Processing Melodic Lines in a Jazz Improvisation.” *International Congress In Music and Artificial Intelligence*, Edinburgh, 103-118.

Joab, M., Paumelle, I., and Delforge, B. (1995). “Architecture du système Diapason.” *95-1*, LIF - Université Pierre & Marie Curie.

Karsenty, S., and Weikart, C. (1994). “Rollit : An Application Builder.” *TOOLS'13*, Versailles, 15-26.

Kautz, H. A., and Allen, J. F. (1986). “Generalized Plan Recognition.” *AAAI '86*, Philadelphia, Pa, 32-37.

Kökeny, T. (1994). “Satisfaction de contraintes dans un environnement orienté objets,” Ph. D., Université Montpellier II., Montpellier.

Kowalski, R. A., and Sergot, M. J. (1986). “A logic-based calculus of events.” *New generation computing*, 4, 67-95.

Krasner, G. E., and Pope, S. T. (1988). “A Cookbook for using the Model-View-Controller User Interface Paradigm in Smalltalk-80.” , ParcPlace Systems.

Krief, P. (1990). “Un système interactif de construction d'environnements de prototypage de multiples outils d'interprétation de modèles de représentation,” Ph.D., Paris 8, Paris.

Laublet, P. (1993). “FORREnMat: un système à base de connaissances pour l'étude expérimentale du raisonnement mathématique,” Ph.D., Université Pierre & Marie Curie.

Laublet, P. (1994). “Hybrid Knowledge Representation and Theorem Proving in Mathematics.” Artificial Intelligence in Mathematics, J. H. Johnson, S. McKee, and A. Vella, eds., Oxford University Press.

Laurière, J. L. (1976). “Un langage et un programme pour énoncer et résoudre des problèmes combinatoires.,” Ph. D, University Pierre et Marie Curie, Paris.

Leman, S., Giroux, S., and Marcenac, P. (1995). “A multi agent approach to modeling student reasoning.” *Artificial Intelligence in Education*, Washington DC, 258-265.

Lenat, D., and Brown, J.-S. (1984). “Why AM and Eurisko appear to work.” *Artificial Intelligence*, 23, 269-294.

Lenat, D. B., and Feigenbaum, E. A. (1991). “On the thresholds of knowledge.” *Artificial Intelligence*, 47, 185-250.

Lenat, D. B., and Guha, R. V. (1990). *Building large knowledge-based systems. Representation and Inference in the Cyc project.*, Addison-Wesley.

Lenat, D. B., Guha, R. V., Pittman, K., Pratt, D., and Shepherd, M. (1990). “Cyc: Towards programs with common sense.” *Communications of the ACM*, 33(8), 30-49.

Liebermann, H. (1986). “Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems.” *OOPSLA '86*, 214-223.

Liebermann, H. (1991). “Habilitation à diriger des recherches. Mémoire de synthèse,” Habilitation, Paris 6, Paris.

Liret, A., Canelas, F., Guigon, J.-M., and Bachellerie, P. (1996). “Alice Talks!” , Université Paris 6, Paris.

Mackworth, A. K. (1977). “Consistency in Networks of Relations.” *Artificial intelligence*, 8(1)(1977), 99-118.

Marinõ, O. (1993). “Raisonnement classificatoire dans une représentation objets multi-points de vue,” Ph. D., Université Joseph Fourier, Grenoble.

McAffer, J. (1995). “A Meta-level Architecture for Prototyping Object Systems,” Ph.D., University of Tokyo, Tokyo.

McDermott, D. (1982). “A temporal logic for reasoning about processes and plans.” *Cognitive science*, 6, 101-155.

Meyer, B. (1992). *Eiffel, the language*, Prentice-Hall.

Micklei, E. (1996). “Tutorial on "Spying messages to objects".” , Summer School of the European Smalltalk User Group, Lausanne, Suisse.

Mili, H. (1988). “Building and Maintaining Hierarchical Semantic Nets,” Doctoral Dissertation, George Washington University, Washington, D.C.

Minsky, M. (1975). “A Framework for Representing Knowledge.” The Psychology of Computer Vision, P. Winston, ed., McGraw-Hill, New York, 211-281.

Minsky, M. (1985). “Musique, sens et pensée.” Quoi ?, Quand ?, Comment ? La recherche musicale, Christian Bourgois, IRCAM, Paris, 137-163.

Minsky, M., and Laske, O. (1992). “A Conversation with Marvin Minsky (Foreword).” Understanding Music with AI, M. Balaban, K. Ebcioglu, and O. Laske, eds., MIT Press, ix-xxx.

Moinard, C. (1994). “Diagral: a diagnosis and repair system.” *Fifth International Workshop on Principles of Diagnosis - DX'94*, New Paltz.

Mouton, R. (1995). "Outils intelligents pour les musicologues," Ph.D., Université du Maine, Le Mans.

Newell, A. (1982). "The Knowledge Level." *Artificial Intelligence*, 18, 87-127.

Newell, A. (1990). *Unified Theories of Cognition*, Harvard University Press.

NKambou, R. (1995). "Modélisation des connaissances de la matière dans un système tutoriel intelligent : modèles, outils et application," Ph.D., Université de Montréal, Montréal.

NKambou, R., Gauthier, G., Frasson, C., and Antaki, M. (1995). "Integrating expert system in authoring systems for curriculum and course building." *7th International Conference on Artificial Intelligence and Expert System Applications*, San Francisco, 485-490.

Ovans, R. (1992). " An Interactive Constraint-Based Expert Assistant for Music Composition." *Ninth Canadian Conference on Artificial Intelligence*, University of British Columbia, Vancouver.

Paquette, G. (1991). "Métaconnaissance dans les environnements d'apprentissage," Ph.D., Université du Maine, Le Mans.

Paquette, G. (1996). "Le Licef : vers de nouvelles technologies d'apprentissage." *Réseau, magasine d'information de l'Université du Québec*, 27(7), 14-19.

Paquette, G., and Girard, J. (1996). "AGD: A Course Engineering Support System." *ITS-96*, Montréal, 382-392.

Perec, G. (1986). *Les mots croisés II*, POL & Mazarine.

Perrot, J.-F. (1995). "Domaines et utilisations de la technologie à objets." *Ingénierie des systèmes d'information*, 3(6), 737-774.

Perrot, J.-F., and Napoli, A. (1996). "Systèmes à objets: tendances actuelles et évolution." *Technique et Science Informatique*, 15(6).

Pierret-Gobreich, C. (1996). "TASK, un environnement pur le développement de systèmes à base de connaissances flexibles," Habilitation à diriger des recherches, Université Paris Sud, Orsay.

Pitrat, J. (1977). "A chess combination program which uses plans." *Artificial Intelligence*, 8, 275-321.

Pitrat, J. (1990). *Métaconnaissances, futur de l'intelligence artificielle*, Hermès.

Pope, S. (1991). "Introduction to MODE: The Musical Object Development Environment." The Well-tempered Object: Musical Applications of Object-oriented Programming, S. Pope, ed., MIT Press, 83-106.

Puget, J.-F. (94). "A C++ implementation of CLP." *Ilog Technical Report*, Ilog, Paris.

Rader, G. M. (1996). "Creating Printed Music Automatically." *IEEE Computer*, 61-68.

Ramalho, G. (1996). "De la construction d'un agent rationnel pouvant jouer du Jazz: une approche à base de connaissances," Ph. D., Université Paris 6, Paris.

Ramalho, G., and Ganascia, J.-G. (1994). "Simulating Creativity in Jazz Performance." *12th AAAI Conference*, Seattle, 108-113.

Ramaux, N., Dojat, M., and Fontaine, D. (1997). "Temporal Scenario Recognition for Intelligent Patient Monitoring." *Artificial Intelligence in Medicine Europe*, Grenoble.

Read, G. (1969). *Musical Notation*, Allyn & Bacon Inc.

Revault, N. (1996). "Principes de méta-modélisation pour l'utilisation de canevas d'applications à objets," Ph. D., Université Pierre & Marie Curie, Paris.

Revault, N., Sahraoui, H., Blain, G., and Perrot, J.-F. (1994). "A Metamodeling Technique: The Metagen System." *TOOLS Europe 16*, Versailles, 127-139.

Roads, C. (1985). "Research in Music and Artificial Intelligence." *ACM Computing Surveys*, 163-190.

Rodet, X., and Cointe, P. (1991). "Formes: Composition and Scheduling of Process." The Well-tempered Object: Musical Applications of Object-oriented Programming, S. Pope, ed., MIT Press, 64-82.

Rolland, P.-Y., and Ganascia, J.-G. (1996). "Automated motive-oriented analysis of musical corpuses: a Jazz case study." *International Computer Music Conference*, Hong-Kong.

Rougegrez, S. (1994). "Prédiction de processus à partir de comportements observés: le système REBECAS," Ph.D., Université Pierre & Marie Curie, Paris.

Rousseaux, F. (1990). "Une contribution de l'intelligence artificielle et de l'apprentissage symbolique automatique à l'élaboration d'un modèle d'enseignement de l'écoute musicale," Ph.D., Paris 6.

Rousseaux, F. (1995). "Contribution à une méthodologie d'acquisition des connaissances pour l'ingénierie des Systèmes d'Information et de Communication: l'exemple de CHEOPS pour l'aide à la gestion de crises collectives à caractère géopolitique," Habilitation, Paris 6, Paris.

Sahraoui, H. (1995). "Application de la méta-modélisation à la génération des outils de conception et de mise en oeuvre des bases de données," Ph.D., Université Pierre & Marie Curie.

Scaletti, C., and Johnson, R. E. (1988). "An Interactive Environment for Object-Oriented Music Composition and Sound Synthesis." *OOSPLA*, San Diego (Ca), 222-233.

Siron, J. (1992). *La partition intérieure*, Outre Mesure.

Smoliar, S. W. (1995). "The Music Collection." *Artificial Intelligence*, 79(2), 341-343.

Steedman, M. J. (1984). "A Generative Grammar for Jazz Chord Sequences." *Music Perception*, 2(1), 52-77.

Steels, L. (1979). "Reasoning modeled as a Society of Communicating Experts." *AI-TR-542*, MIT AI Lab.

Steels, L., and McDermott, J. (1994). *The Knowledge Level in Expert Systems. Conversations and Commentary.*, Academic Press.

Tsang, C. P., and Aitken, M. (1991). "Harmonizing music as a discipline of constraint logic programming." *ICMC '91*, Montréal, 61-64.

Tsang, E. P. (1987). "Times structures for Artificial Intelligence." *10th International Joint Conference on Artificial Intelligence*, 456-461.

Vlissides, J. M., and Linton, M. A. (1990). "Unidraw: A Framework for Building Domain-specific Graphical Editors." *ACM Transactions on Information Systems*, 8(3), 237-268.

Voyer, R. (1989). "Implémentation d'architectures efficaces pour la représentation des connaissances. Application aux langages Loopsiris et OKS," Ph. D., Paris 6, Paris.

Willamowski, J. (1994). "Modélisation de tâches pour la résolution de problèmes en coopération multi-utilisateur.," Ph. D., Université Joseph Fourier, Grenoble.

Wolinski, F. (1990). " Etude des capacités de modélisation systémique des langages à objets appliquées à la représentation de robots," Ph. D., Paris 6, Paris.

Wolinski, F. (1994). "RPC-Talk: une librairie RPC pour Smalltalk." *94/26*, Laforia, Université Paris 6.

Zilberstein, S., and Russel, S. J. (1996). "Optimal Composition of Real-Time Systems." *Artificial Intelligence*, 82(1-2), 181-213.