Un framework pour la représentation de connaissances temporelles par objets et règles

François PACHET[1], Michel DOJAT[2]

[1]LAFORIA-IBP, Université Paris 6, Boîte 169, 4, Place Jussieu, 75252 Paris, France.
[2]INSERM Unité 296, Faculté de médecine, 94010 Créteil, Cedex, France
e-mails: dojat/pachet@laforia.ibp.fr

RESUME. Nous présentons un cadre général pour la représentation de raisonnement temporel dans un univers intégrant programmation par objets et règles de production. Ce cadre est basé sur l'utilisation conjointe de trois mécanismes nés de cette intégration; le typage naturel, l'héritage de bases de règles, et le contrôle déclaratif par méta règles. Ces mécanismes sont utilisés pour représenter deux schémas d'inférence : agrégation et oubli. Le cadre proposé est motivé par l'analyse comparée de deux applications substantielles complètes ayant toutes deux été validées.

ABSTRACT. We introduce a general framework for representing temporal reasoning in an environment integrating object-oriented programming with rule-based programming. The model is based on the simultaneous use of three mechanisms specific to this integration: natural typing; rule base inheritance and declarative specification of control with meta rules. These mechanisms are used to implement two inference patterns deemed essential for temporal reasoning: aggregation and forgetting. The framework is motivated by a comparative analysis of two substantial complete applications that have been validated.

MOTS-CLES: Programmation par objets, raisonnement temporel, règles de production, héritage, patterns, frameworks.

KEY WORDS: Object-Oriented Programming, temporal reasoning, production rules, inheritance, patterns, frameworks.

Un framework pour la représentation de connaissances temporelles par objets et règles

François PACHET, Michel DOJAT

1. Introduction : PPO et RPO

Il existe en France comme à l'étranger un débat de fond sur la différence de nature entre la Représentation par Objets (RPO) et la Programmation par Objets (PPO). Ce débat n'est pas nouveau. Depuis 1988, différents chercheurs ont argumenté sur ce sujet, le plus souvent en comparant les mérites respectifs des deux approches, et en proposant des implémentations de langages de RPO, c'est à dire de frames, par des langages de PPO [Missaoui and Gul 1988; Roche 1989; Rathke 1991]. Ces discussions tournent le plus souvent autour du problème des *facettes* et *valeurs déclenchantes*, et leur implémentation dans un langage de programmation par objets.

Un des initiateurs de cette opposition [Patel-Schneider 1990] a affermi le débat en mettant en évidence les problèmes fondamentaux de représentation posés par l'utilisation de langages de PPO. Parmi ces problèmes on trouve la difficulté de représenter les *valeurs par défauts* et le manque de *sémantique propre* de ces langages. Les langages de RPO sont en revanche considérés comme a priori plus expressifs, mais plus difficiles à implémenter et moins efficaces. Plus récemment, [Carré et al. 1995] ont proposé de mettre en avant le mécanisme de *classification* comme différence principale d'orientation entre les deux courants. Par ailleurs, une proposition intéressante du même papier est de rappeler une des difficultés essentielles de représentation que sont les procédures.

Ce débat est double: il y a un débat sur des différences de *mécanismes*, et un débat sur des différences de *statut* des langages. Nous allons préciser notre position sur chacun de ces points.

Premièrement, si les deux familles de langages ont bien une parenté commune, elles proposent des mécanismes différents, voire incompatibles. Il est vrai que l'héritage de classe n'est pas comparable à un mécanisme de classification (mais qui dit le contraire ?). Il est vrai que les langages de PPO n'ont pas de sémantique formelle précise (pas plus d'ailleurs que les langages de frames), mais quand les langages de RPO purs (logiques descriptives) se dotent d'une sémantique, c'est pour mieux mettre en évidence leur incomplétude [Napoli and Volle 1993]. En ce qui concerne les valeurs par défaut, les représenter sous forme de facettes particulières sur les slots (RPO) ou par des méthodes d'initialisation dans la classe (PPO) ne constitue pas, à notre avis, un véritable thème de débat. De même pour les valeurs déclenchantes, qui correspondent à des méthodes d'accès particulières en PPO. Par ailleurs, le mécanisme de classification est bien plus naturel à mettre en oeuvre dans un langage de frames que dans un langage de PPO, par le fait que la relation d'instanciation y est représentée par un simple slot au lieu d'être enfouie au coeur de l'interprète. Mais il existe bien des moyens d'intégrer un mécanisme de classification dans un langage de PPO

[Yelland

1992]

[Rolland and Pachet

1995]. Sur le problème de la représentation des procédures, en revanche, il nous semble que les langages de PPO ont un avantage, qui est de ne pas essayer de les réduire (sous forme de réflexes), ni de les cacher (sous forme de fonctions Lisp, au statut cabalistique), mais au contraire de les inclure en leur donnant un statut de premier plan, via justement l'héritage de classe, dont la sémantique opérationnelle est donnée par le mécanisme du lookup. Cependant, il est vrai que, dans un cas comme dans l'autre, les procédures forment un noeud que l'on ne sait que contourner, plus ou moins habilement.

Deuxièmement, il y a le problème du statut des langages. Pour des raisons obscures, les partisans de la RPO attribuent à leurs langages des vertus *modélisatrices* (*représentatoires ?*) que n'auraient pas les langages de PPO. La présence d'un mécanisme particulier (classification, valeurs déclenchantes, etc.), si complexe soit-il, ne permet pas à notre avis d'élever un langage au rang de langage de *modélisation.* Ou alors, si c'est le cas, il faudrait préciser la nature de ces mécanismes "sanctificateurs". En revanche, la nécessité de disposer d'un langage de modélisation est évidente, à la fois pour réfléchir sur la représentation de connaissances de manière indépendante d'un langage, et pour expliquer ou maintenir des bases de connaissances complexes. Mais sur ce point important nous ne considérons pas les langages de RPO comme plus appropriés que les autres pour remplir ce rôle délicat. Nous suivons plutôt la voie engagée pour les langages de PPO avec les travaux sur les "patterns de conception" (*design patterns*) et les *frameworks,* qui, bien qu'en cours d'élaboration, nous paraissent capables de proposer des structures abstraites et relativement indépendantes d'un langage particulier.

Comme le souligne [Schank 1991], nous pensons que ".. the ability of a AI program to scale-up is critical" et que donc le véritable problème derrière ce débat est la capacité de ces formalismes à supporter la modélisation et la construction effective de *gros systèmes* à base de connaissances, intégrant des mécanismes différents de représentation et d'inférence.

Pour ce faire, la technologie offerte par les langages de PPO a fait ses preuves. Celle des langages de RPO, plus ancienne, n'a pas été conçue pour permettre la collaboration, la réutilisation, et la modularité. Interrogé sur la raison du manque d'impact des technologies d'Intelligence Artificielle dans la communauté du génie logiciel, R. Balzer

[Balzer 1990] identifie 8 raisons principales: les langages idiosyncratiques, les environnements idiosyncratiques, les hardwares idiosyncratiques, les infrastructures insulaires, les exigences irréalistes en mémoire centrale, le manque d'interopérabilité, la mentalité égocentrique, et l'absence d'encapsulation. Sur ce dernier point, l'encapsulation, la remarque de R. Balzer, qui date de 1990, particulièrement significative:

> ".. In the pursuit of flexibility, we have largely ignored and failed to support the mechanisms used by others for early binding and declarations of static structure. We have no interface definition languages, encapsulation mechanisms or type-safe languages. In the absence of such encapsulation mechanisms, we have no reliable means of breaking systems up into well-defined pieces. Hence the understandability and maintainability of our systems is highly suspect, relying instead on conventions, good coding styles, and accurate documentation".

Malgré ses limitations intrinsèques maintenant bien connues, la programmation par objets est pour nous un paradigme qui mélange effectivement les deux niveaux implémentation et représentation. Loin d'être une limitation, cette confusion de niveaux permet la construction de bases de connaissances substantielles, et ainsi d'atteindre le fameux "scale-up" caractérisitque des systèmes d'Intelligence Artificielle. Dès lors, notre position consiste à introduire les mécanismes réputés d'IA dans un langage de PPO lorsqu'ils s'avèrent effectivement adaptés à une partie de la résolution. Cette adjonction se fait de manière à respecter autant que possible les principes de la PPO, l'encapsulation et l'héritage de classe en particulier, que nous considérons comme les garants du bon fonctionnement global du système, et de sa capacité à être effectivement utilisé dans des applications de grande taille.

C'est cette position qui nous a conduit à étudier l'intégration de mécanismes de règles de production dans un langage de PPO [Pachet 1992a]. Nous avons également étudié l'intégration de mécanismes de satisfaction de contraintes

[Pachet and Roy 1995], de reconnaissance de plans d'actions [Paquette et al.

1995], et plus récemment la classification par les logiques de description (couplage de Smalltalk et BACK, [Rolland and Pachet 1995]).

C'est en poursuivant dans cette direction que nous proposons, dans ce papier, un cadre général pour la représentation de *connaissances temporelles hiérarchiques*. Ce cadre est issu de deux expériences substantielles de fabrication de systèmes opérationnels, complets, et validés. Les deux systèmes ont été développés à l'aide de techniques de PPO (Smalltalk) et de règles de production (NéOpus). Ils résolvent deux problèmes totalement différents : le contrôle de la ventilation artificielle de patients dans les unités de soins intensifs d'une part, et l'analyse automatique de séquences d'accords de l'autre. Ils ont été développés de manière indépendante, mais ont produit deux architectures en beaucoup de points semblables. Ils constituent deux exemples particulièrement représentatifs d'utilisation de nos techniques et de mise en application de notre philosophie générale concernant la fabrication de bases de connaissances complexes et opérationnelles par les techniques de PPO.

Le reste du papier est structuré comme suit. Nous présentons d'abord notre contexte de représentation, le système NéOpus, en le replaçant dans la perspective générale de l'intégration objets et règles de production. Nous détaillons en particulier les trois mécanismes nés de cette intégration, qui forment la clé de voûte de notre cadre de représentation. Nous présentons ensuite le premier modèle de raisonnement temporel construit au dessus de NéOpus, et exploitant pleinement ces trois mécanismes. Puis, nous présentons le deuxième système de raisonnement, pour l'analyse de séquences d'accords. Enfin, nous discutons et comparons les deux systèmes en montrant ce qu'ils nous enseignent sur l'utilisation conjointe de la PPO et de la programmation par règles pour la représentation de connaissances temporelles. Nous replaçons en particulier notre travail dans le cadre des *patterns* et des *frameworks*.

2. Des objets et des règles

Le besoin d'intégrer des objets et des règles de production a été reconnu depuis longtemps. La base de faits d'un système de règles est un modèle d'une situation concrète modélisée. Afin de donner un peu de sémantique à ces faits, ceux-ci peuvent être considérés comme des propriétés d'*objets constituant* le monde modélisé. Les faits individuels ne sont alors plus représentés comme tels, et leur valeur logique est déduite de requêtes sur les objets dont ils sont une des propriétés. Ainsi, on parvient à dissoudre la base de faits dans le modèle objets du domaine. Cette opération est si naturelle, que les objets sont apparus dans les systèmes à base de règles depuis le système OPS-5.

L'inférence en chaînage avant peut effectivement être considérée comme une extension naturelle de la programmation par objets ordinaire. L'exécution d'un programme objets effectue une série de modifications d'un ensemble d'objets. De la même manière, l'exécution d'un programme de règles en chaînage avant est un cycle de modifications de la base de faits (on parle de programmation dirigée par les données). Ainsi, l'équation "base de faits = modèle du monde" relie les deux techniques.

Si le mouvement est naturel, l'équation ne satisfait pas tout le monde. En particulier, le style de programmation par envoi de message est considéré par les spécialistes de représentation de connaissances comme une violation des principes de déclarativité, Cf.

e.g. [Nebel 1990]. Les objets apparaissent comme des entités fermées accessibles seulement de l'extérieur, par leur interface, définie dans leur classe. L'obtention des propriétés logiques se fait alors par l'intermédiaire de procédures, explicitement définies dans la classe, alors que dans un modèle à frames, le système a un accès direct aux slots, supposés contenir plus de sémantique. Notre expérience de la PPO comme couche de représentation de connaissance pour la fabrication effective de systèmes d'IA nous a conduit à penser que la possibilité de réutiliser le travail des autres valait bien quelque entorse à un dogme établit.

2.1. État de l'Art

De nombreux systèmes ont été produits pour proposer une intégration objets et règles, et diffèrent essentiellement soit par le mécanisme de compilation de règles choisi

[Pachet 1994c], soit par le langage objet "racine" utilisé.

En ce qui concerne le mode de compilation des règles, il y a essentiellement deux tendances: la compilation issue de RETE

[Forgy

1982], et la compilation par chemin (*path-based production rules*) [Crawford and Kuipers 1991; Crawford et al. 1994]. Dans les deux cas, le principal problème technique est celui dit du *modified* dans la terminologie NéOpus, appelé aussi *behavioral match problem* ou *notification problem*.

Ce problème est non résolu dans sa généralité, et chaque système propose une approche différente.

Du point de vue du langage objet autochtone utilisé, on retrouve encore la distinction RPO/PPO. Pour les langages de frames, l'effort le plus visible est celui de CLIPS, qui associe à un langage de frames bâti au dessus de C (COOL), à un langage de règles inspiré directement d' OPS-5. CLIPS est une entreprise remarquable par la quantité de travail qu'elle représente (NASA), et le nombre important d'applications qui ont été réalisées, et qui ont fait l'objet de descriptions largement publiées (3 conférences spécialisées à ce jour, recensant plusieurs centaines d'applications, Cf. par exemple [CLIPS 1994]). Néanmoins, le problème de la communication entre le langage objet de CLIPS et le langage objet source (C++) est épineux et pour l'instant non réglé (Cf. e.g. les problèmes de cohabitation entre COOL et C++ dans [Dvorak 1994]). Il rend particulièrement problématique la réutilisation effective de systèmes non conçus pour être intégrés dans le système de règles.

Les tentatives d'intégration dans des langages de PPO sont plus diverses. On peut citer les travaux de Forgy (RAL/C++) [Forgy 1994], l'initiateur de la compilation RETE, et de Miranker [Miranker

1994], centrés essentiellement sur des problèmes d'efficacité (ils proposent des systèmes 10 fois plus rapides que CLIPS). En France, le système IlogRules [Albert 1994], est lui aussi issu de la technique RETE, et adopte une politique de conformité maximum au langage C++. Il faut noter que les deux principaux auteurs de la compilation RETE (C. Forgy, D. Miranker) s'intéressent aujourd'hui à l'intégration de leur mécanisme dans un langage de PPO (C++ en l'occurrence) et non dans un langage de RPO [Pachet 1994c]. Dans la même tradition, nous avons construit le système NéOpus, décrit dans le section suivante.

2.2. NéOpus

Notre contexte de représentation est le système NéOpus. Du point de vue technique, il s'agit de l'intégration de mécanismes de compilation de règles d'ordre un par réseau RETE en Smalltalk. NéOpus a été utilisé dans de nombreuses applications, et est aujourd'hui distribué par ftp[1]. L'architecture de NéOpus a par ailleurs été adoptée pour le système ENVY/Expert développé par la société *Object Technology International*, qui lui a adjoint la technologie de programmation par équipe ENVY, en faisant un des seuls système à ce jour permettant la programmation de grosses bases de règles partageables

[1] site ftp anonymous ftp.ibp.fr, directory softs/Laforia/NeOpus

[Barry and McAffer 1992].

Un des points importants concernant l'intégration de règles et d'objets est la possibilité d'écrire des règles sur des objets quelconques, sans imposer de contraintes sur leur superclasse. Ceci est le garant de l'utilisation effective d'applications et de librairies de classes au sein d'un système à base de connaissances complexe.

Du point de vue de la représentation de connaissances, cette attitude revient à considérer que tout programme peut être vu comme une forme particulière (maladroite) de représentation de connaissances. Néanmoins, quand le langage d'implémentation est un langage de PPO, les catégories utilisées par le programme (classes, instances, méthodes) deviennent alors plus accessibles et peuvent se prêter à des manipulations automatiques en vue d'être effectivement utilisées par des programmes de raisonnement.

D'un point de vue technique, le fait que chaque objet possède une interface de communication définie par sa classe peut être tourné en un avantage en imposant que les règles soient exprimées entièrement en terme d'expressions du langage objet (envois de messages) sans construction syntaxique particulière (contraintes sur attributs). Les conditions des règles sont alors exprimées par des messages à valeur booléenne, et les actions par des messages à effet de bord (procédures) quelconques.

2.3. Trois mécanismes clé

La représentation de connaissances en NéOpus repose ainsi sur l'utilisation conjointe de structures objets Smalltalk et de règles de production. Bien sûr, cette intégration est plus que la somme de ses parties. Notre expérience du système a fait émerger trois mécanismes principaux, nés de cette intégration: 1) le typage naturel, 2) l'héritage de bases de règles et 3) l'architecture déclarative de contrôle. Nous allons les décrire brièvement dans les sections suivantes.

2.3.1. Typage naturel et héritage de classe

L'héritage de classe est un mécanisme clé de la PPO sur lequel il est inutile de revenir. Un des aspects importants de l'intégration règles et objets est la prise en compte de l'héritage de classe dans le mécanisme de filtrage des règles de production. Le principal problème à résoudre est en effet le statut des variables dans les règles, et la manière dont les sous-classes des classes de déclaration des variables sont prises en compte. NéOpus propose deux solutions: le typage dit "simple" et le typage "naturel"[2]. Dans le typage simple, les variables sont instanciées par les instances directes de leur classe de déclaration. Dans le typage naturel, les variables sont instanciées par les instances de leur classe de déclaration ET des éventuelles sous-classes.

Le typage naturel est de fait un mécanisme puissant puisqu'il permet de factoriser l'écriture de règles. L'idée du typage naturel est d'autoriser le mécanisme de filtrage à considérer les instances directes et indirectes des classes de déclaration des variables. Ceci permet de rendre l'interprétation de la règle *dynamique*, puisque les parties condition et action des règles sont définies par des envois de messages, et que ces messages sont - par définition de l'héritage de classe - redéfinis dans les sous-classes et recherchés dynamiquement. En d'autres termes, ce mécanisme permet de rendre les règles dépendantes du contexte, le contexte étant représenté par les n-uplets qui

[2] Les autres systèmes intégrant objets et règles proposent en général des mécanismes analogues

instancient la règle [Dojat and Pachet 1995].

Plus précisément, soit r une règle déclarant n variables v_i (i=1, n). Soient C_i, les classes de déclaration des variables vi. Si chaque C_i a k_i sous-classes concrètes, le nombre total d'interprétations différentes de la règle est $\prod k_i$ (i = 1, n). En pratique, si r a 3 variables et chaque classe de déclaration a 5 sous-classes concrètes (cas fréquent), la règle a $5^3 = 125$ interprétations différentes possibles (Cf. Figure 1).

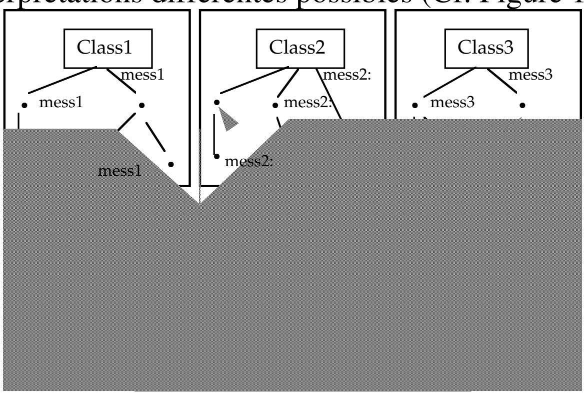

Figure 1. *Une règle avec les hiérarchies de classes correspondantes. La règle se lit: "Pour tout v1, instance de la classe C1, v2 instance de C2 et v3 instance de C3, si "v1 mess1" et "v2 mess: v3" rendent vrai, alors évaluer "v3 mess3". La règle utilise trois messages (mess1, mess2: et mess3) qui sont définis et redéfinis dans les sous-classes des classes Class1, Class2 et Class3.*

2.3.2. L'héritage de bases de règles

L'Héritage de Bases de Règles (HBR) est un mécanisme né d'une observation simple: les bases de règles en NéOpus étant représentées comme des classes abstraites, le problème du statut de l'héritage pour ces classes se pose naturellement. Nous avons introduit un mécanisme permettant de transposer *l'intuition* de l'héritage de classe (spécialisation, masquage et redéfinition) dans le monde des règles [Pachet 1992b]. Ce mécanisme s'est avéré en pratique essentiel pour maîtriser la complexité de bases de règles, et favoriser leur réutilisation.

Grâce au mécanisme d'héritage, chaque base de règle peut être définie comme une *sous-base* d'une base existante. Dans ce cas elle hérite des règles de sa super base, et récursivement des super bases de sa super base. Un mécanisme de masquage est rajouté, qui permet de redéfinir localement une règle héritée, de manière analogue au masquage de l'héritage de classe. L'héritage ainsi que le masquage sont implémentés de manière statique, pour des raisons d'efficacité, a l'aide d'un algorithme de propagation des compilations de règles dans les sous-bases (l'algorithme n'est pas trivial car il faut aussi prendre en compte les suppressions de règles, avec re-propagation de règles anciennement masquées, Cf. [Pachet 1992a] pour les détails).

Par ailleurs, nous avons montré que le mécanisme de lookup, essentiel pour l'interprétation de l'héritage de classe, n'avait pas d'équivalent direct possible dans un monde de règles de production, puisque les règles ne sont pas - par définition - appelées explicitement. Nous avons proposé de remplacer le lookup pour les bases de règles par une stratégie de contrôle partielle, appelée stratégie HBR, et qui consiste à préférer, dans les cas de conflit, les règles les plus spécifiques, c'est à dire définies dans les sous-bases les plus profondes dans l'arbre d'héritage. Cette stratégie est cette fois dynamique. Elle est intéressante dans la mesure ou elle permet de *réduire* le problème du contrôle, sans toutefois lui apporter une solution définitive, les conflits potentiels entre règles d'une même base n'étant pas résolus.

2.3.3. *L'architecture déclarative de contrôle par méta-règles*

Le problème du contrôle dans les systèmes de règles en chaînage avant est crucial [Davis 85]. NéOpus propose une architecture pour le contrôle [Pachet and Perrot 1994] qui repose sur le principe de réflexivité suivant: le contrôle de l'exécution d'une base de règles est vu comme un problème qui nécessite une certaine forme de connaissances. Pour le régler, on utilise NéOpus lui-même sans construction particulière, avec en particulier les deux mécanismes précédents (typage naturel et héritage de bases de règles). Ainsi, l'exécution d'une base de règles est entièrement représentée par l'exécution d'une *autre* base de règles (appelée sa méta-base). Les méta-bases sont des bases de règles ordinaires. La principale différence est que les méta-bases manipulent des "objets de contrôle", interface entre les méta-règles et les éléments vitaux des bases de règles (règles déclenchables, conflict set). Chaque base de règles peut être soit contrôlée par une méta-base, soit contrôlée de manière procédurale (Cf. Figure 2). L'empilement de niveaux méta est ainsi entièrement décidé par l'utilisateur.

Cette architecture a comme particularité principale de permettre la réutilisation effective de spécifications de contrôle, via en particulier l'héritage de bases de règles.

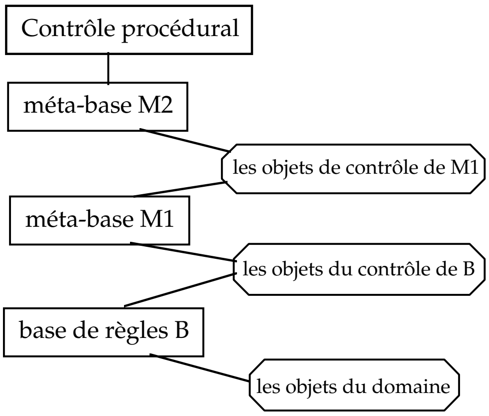

Figure 2. *L'architecture déclarative de contrôle en NéOpus.*

2.4. Représentation de connaissances temporelles

Dans bon nombre d'applications, le raisonnement a une composante temporelle. La représentation du temps est un vaste sujet et il s'agit pas de le résoudre dans sa totalité. Le type de raisonnement temporel que nous voulons représenter ici est un raisonnement dans lequel on a besoin d'exprimer des connaissances sur des *configurations temporelles hiérarchiques*, à plusieurs niveaux d'abstraction.

Nous allons présenter deux applications réalisées en NéOpus, qui font intervenir toutes deux une certaine forme de raisonnement sur le temps. Nous allons montrer pour chacune d'elle, ce que l'environnement de représentation NéOpus a apporté, et en particulier comment les trois mécanismes ont été utilisés. Nous discuterons ensuite de la généralisation possible de ces deux expériences.

3. Un modèle du temps pour le monitorage temps réel : le Système NéoGanesh

NéoGanesh est un système pour le monitorage intelligent en temps réel de patients hospitalisés dans des unités de soins intensifs. Il est appliqué à la surveillance de la fonction respiratoire. Il fonctionne en boucle fermée, en appliquant les actions thérapeutiques qui résultent de ses déductions directement sur le respirateur [Dojat and Pachet

1992]. NéoGanesh, élaboré avec des médecins réanimateurs et des physiologistes (INSERM U296), est utilisé à l'Hôpital Henri Mondor. Deux études cliniques [Dojat et al. 1992; Dojat et al. 1995] ont montré que 1) le système maintenait le patient dans un état de "confort ventilatoire" en adaptant continuellement le niveau d'assistance selon les besoins physiologiques du patient et 2) la valeur prédictive positive du diagnostic effectué par le système, quant à la capacité du patient à respirer sans assistance, est meilleure que celle obtenue à l'aide des tests standards pratiqués (tests de présevrage + pièce en T). Une évaluation de NéoGanesh à plus grande échelle (projet européen BIOMED) est en projet, dans le but de valider cliniquement le système et d'étudier sa possible intégration dans la carte de contrôle du respirateur.

3.1. Le Contexte Médical

Le contexte est le suivant : un réanimateur observe un patient ventilé artificiellement et effectue des actions sur le respirateur pour maintenir le patient dans un état ventilatoire jugé acceptable. Le patient est ventilé en aide inspiratoire, mode de ventilation où le patient déclenche l'aide mécanique lors de l'inspiration et la stoppe lors de l'expiration.

Nous avons trois phases thérapeutiques distinctes : la thérapie dite "standard" (T1), "présevrage" (T2) et "sevrage" (T3). Ces trois phases s'enchaînent chronologiquement avec l'amélioration progressive de l'état ventilatoire du patient. Si l'état du patient se dégrade, le médecin va devoir revenir à une phase thérapeutique antérieure tout en tenant compte des résultats déjà obtenus. par exemple, sous certaines conditions la phase de présevrage pourra être omise si le patient a été déclaré sevrable avant une subite détérioration de son état.

Les actions effectuées sur le respirateur pour maintenir le patient dans une zone de confort respiratoire sont adaptées selon la phase thérapeutique en cours. Ainsi, si le patient est déclaré sevrable et que sa ventilation se dégrade, l'assistance sera diminuée par paliers dès qu'il aura de nouveau une ventilation satisfaisante.

Quatre types d'action sont envisagées (maintien, restauration, réduction et alarme). En fonction de ses actions, le médecin s'attend à ce qu'une nouvelle situation apparaisse (*état souhaité* par exemple, maintien, restauration) en continuation ou en rupture avec l'évolution naturelle (*état prévu*) de la situation existante. Si la thérapie mise en place n'a pas les effets souhaités, il y aura soit adaptation si les effets sont notables mais insuffisants, soit changement si les effets sont nuls voire néfastes.

Un scénario typique est le suivant : le clinicien diagnostique une ventilation insuffisante (le volume inspiré est faible). Durant les 15 dernières minutes, deux diagnostics de ventilation correcte ont été effectués et cette situation aurait du persister. Cette ventilation insuffisante représente une rupture qui persistera si aucune action n'est réalisée. L'assistance mécanique est augmentée et le clinicien prévoit une amélioration de la ventilation. Si la prévision se réalise, l'épisode de ventilation insuffisante pourra être considéré comme une instabilité locale. La ventilation sera alors considérée comme globalement stable. L'assistance pourra être diminuée à nouveau.

3.2. État de l'art sur le raisonnement temporel

Le contrôle de systèmes dynamiques impose une représentation explicite du temps et une modélisation d'un raisonnement temporel. Le formalisme temporel choisi doit permettre à la fois de raisonner sur des événements évolutifs, de prendre en compte des événements exceptionnels, de gérer l'historique et de prédire les situations futures.

Divers travaux en IA sont consacrés à la représentation du temps. Ils peuvent être regroupés en fonction du type de primitive temporelle sur laquelle ils reposent :

intervalle [Allen

1984], *points*

[McDermott

1982] ou *événements* [Kowalski and Sergot 1986]. Le choix effectué reflète la dimension du temps que l'on considère comme fondamentale : la durée (intervalle) ou l'instant (point, événement). Formellement, les trois approches peuvent être reliées les unes aux autres et permettent de représenter les mêmes connaissances [Tsang 1987].

Généralement, les travaux théoriques proposés insistent plus sur la manière de *raisonner* sur le temps que sur la *représentation* des connaissances temporelles. Ils s'attachent principalement à définir des méthodes pour propager des contraintes temporelles dans un graphe, dont les nœuds sont les primitives de base, points ou intervalles, et les arcs les relations entre ces primitives. Or, dans le cas d'applications réelles les situations temporelles s'avèrent variées et complexes et nécessitent des structures temporelles multiformes, adaptables au discours à représenter

[Sayettat 1990]. Un décalage important existe entre la qualité des travaux théoriques en logique, en informatique, et en IA pour la représentation d'un raisonnement temporel, et la rusticité des outils opérationnels. En particulier, malgré différents travaux, (voir sur l'algèbre d'instants

[Dechter et al.

1991]

[Van Beek

1992]), la complexité des algorithmes de propagation de contraintes temporelles limite l'intégration des gestionnaires temporels dans des systèmes temps réels (NP complet sur une algèbre d'intervalles et polynomial sur une algèbre d'instants ou algèbre d'intervalles restreinte

[Vilain and Kautz 1986]).
Des approches plus pragmatiques ont été proposées [Lau and Vincent

1993]

[Williamson and Hanks

1993]. Elles sont moins générales mais plus efficaces parce qu'elles exploitent les connaissances et la structure du domaine considéré. Nous poursuivons dans cette direction.

3.3. Le modèle de représentation temporelle: agrégation et oubli

Bien que l'enchaînement des faits soit d'une grande importance en médecine, la composante temporelle est une caractéristique rarement abordée ou mal formalisée dans la plupart des systèmes médicaux à base de connaissances. Ces systèmes possèdent une image figée du patient à un moment précis et prennent rarement en compte l'évolution de la maladie [Shortliffe 1990]. Ils considèrent que les symptômes décrits sont présents depuis une durée spécifiée et établissent un diagnostic à un instant t.

Notre approche est basée sur le point de vue philosophique suivant : le temps existe parce que nous percevons le changement. La perception du changement est intimement liée au contexte, aux actions que l'on envisage d'effectuer sur le monde. Lors du monitorage de la ventilation d'un patient, le médecin est intéressé par l'évolution des déséquilibres physiologiques, par la réponse du patient à certaines actions. Il cherche les réponses à des questions telles que : "depuis combien de temps le patient est-il hyperventilé ?", "l'augmentation de l'assistance mécanique a-t-elle été efficace ?" "le patient est-il stable depuis assez longtemps pour pouvoir diminuer l'assistance ?". Un modèle construit sur un ensemble d'événements et d'états, muni de relations de cause à effet entre eux, paraît bien adapté à nos besoins

[Dojat and Sayettat 1995].

Notre modèle de raisonnement temporel est basé sur deux mécanismes : *l'agrégation* et *l'oubli* d'informations. Ces deux mécanismes requièrent des connaissances qui dépendent du domaine et du contexte d'utilisation. Ils permettent de construire de façon incrémentale une interprétation de l'évolution de l'état du patient en limitant la propagation des contraintes temporelles aux besoins du discours à modéliser.

3.4. Ontologie Temporelle

Notre ontologie divise le monde entre 1) les entités "existantes" du domaine, entités atemporelles, qui par essence n'ont aucune dimension temporelle et permettent de décrire le monde et 2) les entités marquées d'un sceau temporel (entités temporelles) utilisées pour développer un discours sur les changements du monde. Ces entités sont considérées comme des entités conceptuelles sur lesquelles un opérateur peut avoir des connaissances et avec lesquelles il peut raisonner.

Chaque entité temporelle est repérée sur un axe temporel et définira une information (objet, propriété d'un objet) valide pour une période (notion proche des "time structures" de [Murray and Balaban 1988], ou des "contexts" de [Hewitt

1971]). Ainsi, nous utilisons un formalisme unitaire, de type <t,p> où t est une marque temporelle (instant ou intervalle) et p une assertion atemporelle.

Les entités atemporelles constituent les entités du domaine à modéliser et existent préalablement à tout discours temporel. Elles ne connaissent que leur état présent. Aucune notion de durée ne leur est accessible. Elles subissent des changements mais ne sont pas concernées par leur propre évolution. Dans un environnement médical, ce seront par exemple, un paramètre physiologique, une pathologie, un moniteur de gaz carbonique. Tous ces objets sont différents par nature mais non pas été différenciés pour leur comportement temporel. Ils seront instances de n'importe quelle classe préexistante.

Nous considérons le temps comme un ensemble ordonné et linéaire d'instants. Une métrique sera disponible à l'aide de dates, marques précises sur un axe temporel isomorphe à la droite des réels, exprimables dans diverses unités. La résolution (ou granularité) dépend de l'application. Nous raffinons la notion d'entités temporelles en trois concepts : `Evenement`, `Etat`, et `Chronique` qui se distinguent par leur comportement dynamique.

3.4.1. La Notion d'Événement

Un événement a une dimension temporelle et son apparition vient modifier l'état du monde. Comme chez [Kowalski and Sergot 1986] la notion d'événement est centrale dans notre approche. L'événement peut introduire un *changement* dans notre représentation du monde (initie ou stoppe un objet temporel). Généralement, il occupe l'espace temporel de façon ponctuelle ou bien sa durée est sans intérêt. Nous pouvons lui associer une marque temporelle correspondant à la date de son apparition.

3.4.2. La Notion d'État

L'état introduit la notion de durée. Un état caractérise une propriété locale du monde qui remplit l'espace, une information qui dure pour une période de temps. Les états sont chaînés continûment. La transition entre états successifs est causée par l'apparition d'un événement et un état est assuré de persister tant qu'aucun événement ne le termine. Nous considérons que les transitions sont instantanées, qu'elles ont lieu à un instant précis déterminé par l'événement initiateur. Un état possédera donc une période de validité, représentée par un intervalle borné par des événements qui correspondent à son début et éventuellement à sa fin. L'état est identique à la notion de "State" chez

[McDermott 1982], de propriété qui dure (prédicat Hold chez [Allen

1984]), ou de proposition chez [Kowalski and Sergot 1986]. L'information représentée par un état est persistante sur la période de temps associée, soit un intervalle. Par défaut, l'information valide sur un intervalle est valide sur les sous-intervalles. Nous avons introduit la notion d'intervalle discontinu pour lequel une information n'est pas persistante sur tous les sous-intervalles qu'il contient[3].

[3]Notion similaire aux processus de Allen.

3.4.3. La Notion de Chronique

Définie comme un ensemble d'objets temporels se succédant, une chronique conserve les objets temporels pérennes (par exemple les états ventilatoires du patient, les actions sur le ventilateur, ...). Le terme chronique n'a pas ici le même sens que chez [McDermott 1982] puisqu'il signifie non pas une histoire possible du monde mais "la suite des faits consignés dans l'ordre de leur déroulement", comme l'indique la définition française du mot[4]. Les objets doivent être stockés sous une forme exploitable en évitant perte d'information et redondance. Pour des raisons de rapidité d'accès aux informations, les données courantes sont séparées des données historiques. Les chroniques constituent une base de données temporelles.

3.5. Les Inférences Temporelles

Beaucoup de travaux sur le raisonnement temporel sont effectués pour améliorer l'efficacité des algorithmes de propagation de contraintes, afin de manipuler de grandes

[4] Nous n'introduisons pas de différence entre le moment où une information est enregistrée (belief time) et le moment où elle est le reflet de la réalité (historical time). Nous considérons qu'une information est enregistrée au moment même où elle correspond à la réalité.

quantités de données avec des performances acceptables. Cependant, face à des problèmes réels, ces approches restent difficiles à utiliser et des considérations dépendantes du domaine doivent être introduites. C'est pourquoi notre solution est calquée sur l'attitude de l'opérateur qui réalise une tâche de surveillance.

3.5.1. Les Abstractions Temporelles

Dans le cas du monitorage en continu d'un procédé, de nombreuses données sont accumulées au cours du temps. L'opérateur qui surveille le système est capable, après avoir diagnostiqué l'état courant, 1) de comparer et de regrouper les différentes situations afin d'apprécier l'évolution du système, 2) de comparer cette évolution avec des scénarios typiques et avec celle attendue, 3) d'élaborer des plans d'actions et de nouvelles prévisions d'évolution. Ces tâches sont structurées temporellement [Dojat and Sayettat 1995] ; la construction d'un graphe temporel n'est pas nécessaire. Les observations sont réalisées à des instants précis et sont totalement ordonnées. Nous considérons que l'expert réalise trois formes d'abstractions temporelles : *création de relations événements-états*, *agrégation* d'informations et *oubli* d'informations (voir figure 3).

3.5.1.1. Relations Evénements-Etats

L'apparition d'un événement peut terminer la période de validité associée à un objet temporel ou créer de nouveaux objets temporels. Il y a une relation de cause à effet entre les événements et les états. Les événements ne sont pas nécessairement imprévisibles. Certains événements sont déclenchés pour agir sur le monde ou pour apprécier un changement local en réalisant une scansion du temps.

Lors de la surveillance du patient, après l'étape de classification de la situation observée, basée sur des heuristiques propres au domaine, la proposition d'un diagnostic introduit une nouvelle explication à un instant donné de l'état du malade. Les observations initiales *quantitatives, réalisées à un instant t,* sont converties en informations *qualitatives valides sur un intervalle de temps*. Nous considérons que par *interpolation* la situation reconnue dure au maximum depuis le dernier diagnostic. Nous réalisons le passage d'une représentation à l'aide de point (événement) à une représentation à l'aide d'intervalle (état continu).

3.5.1.2. Agrégation et Oubli

Au cours du temps, plusieurs états successifs peuvent être diagnostiqués et accumulés. Mais guidé par la nécessité naturelle d'agir, l'expert agrège les états *similaires*, repère les situations de rupture et oublie les informations redondantes ou obsolètes. Agrégation et oubli sont utilisés pour mettre dynamiquement en relief certaines informations utiles au raisonnement en cours, et estomper celles qui n'intéressent pas l'action présente. L'activation de ces deux mécanismes dépend du contexte et permet de créer dynamiquement de nouvelles descriptions du comportement du système et d'élaborer des plans d'actions [Dojat and Sayettat 1995].

Les inférences sont réalisées à partir d'informations temporelles valides à la fois sur des intervalles contigus et disjoints. Il s'agit essentiellement de gérer le changement d'état (rupture) ou d'agréger des états considérés comme similaires, qui reposent soit sur des informations réelles (le présent) soit sur des informations fictives (les prévisions).

Sous certaines conditions, un mécanisme d'oubli est déclenché qui élimine des connaissances redondantes ou superflues de l'espace de raisonnement courant. Le mécanisme d'oubli est primordial pour les systèmes à mémoire naturels ou artificiels. Il y a plusieurs manières d'oublier. Nous introduisons deux formes sommaires qui ne réalisent pas de disparition d'information du système, mais un refoulement volontaire ou involontaire :

1) Un oubli actif : il s'agit d'une *action volontaire*. Des informations sont délibérément éliminées de l'espace de raisonnement au cours du processus de réflexion.

2) Un oubli passif : Ce mécanisme est *déclenché automatiquement*. Des informations qui ne sont pas utilisées pendant une période de temps s'évanouissent. La quantité d'informations présente dans la mémoire à court terme est maintenue au-dessous d'un seuil.

3.5.2. *La perception du changement*

Juger que deux objets temporels sont différents nécessite de définir en quoi l'information qu'ils représentent a été modifiée au cours du temps. Ce jugement est liée à la sémantique de l'application et devra être précisé au moment de la mise en place de la base de connaissances.

Lors de la création d'une classe d'objets temporels nous définirons les caractéristiques pertinentes, en quelque sorte l'essence des instances, qui devront être conservées pour que deux instances soient identiques. Les sous-classes de `ObjetTemporel`, en redéfinissant la notion d'identité entre deux instances, refléteront en quelque sorte d'autres visions du changement adaptées au contexte.

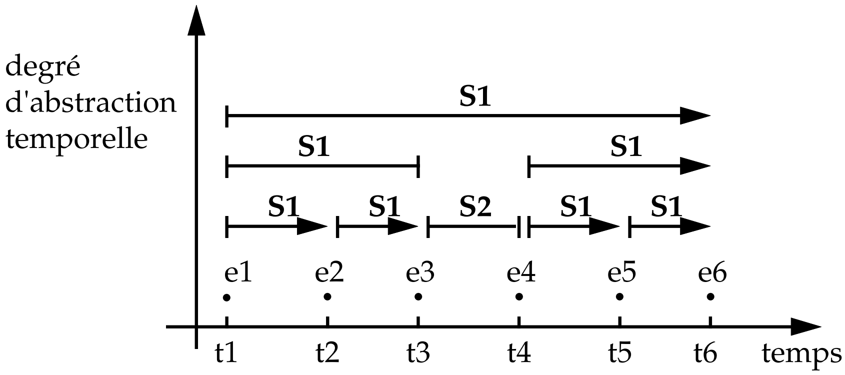

Figure 3. *Abstractions temporelles*
A différents instants t1, ..., t5, des diagnostics sont effectués (événements e1, e2, ..., e6) et une situation est reconnue (état S1). Un état est assuré de durer au moins sur la période séparant deux diagnostics. Un état terminé est représenté par un intervalle borné. Un état persistant est représenté par un intervalle semi-borné. Lorsque deux situations similaires sont reconnues, il a y agrégation. Une situation différente (état S2) peut provoquer une rupture qui, si elle est de courte durée, peut être oubliée. La pérennité de l'état S1 est alors maintenue.

3.6. Représentation en NéOpus

Le modèle que nous avons présenté dans le section précédente, les structures et les abstractions temporelles, sont représentées en utilisant NéOpus, et en particulier les trois mécanismes présentés au début de ce papier. Nous avons de plus raffiné notre modèle temporel initial pour prendre en compte les particularités des situations cliniques.

3.6.1. *Représentation des objets temporels*

A partir des concepts temporels de base nous voulons pouvoir définir des concepts plus spécifiques pour la modélisation de divers discours temporels et ainsi incorporer, par *redéfinition* et *raffinement* des notions existantes, la sémantique utile à une application. Afin de mettre en place ces différents niveaux, nous proposons une organisation hiérarchique définissant une typologie des connaissances avec une spécialisation graduelle des propriétés et des contraintes.

L'objet de base de notre langage de représentation est appelé `ObjetTemporel`. Un objet temporel (OT) est un objet complexe qui possède deux attributs : une `marque` qui le situe sur un axe temporel et une `valeur` qui référence une entité atemporelle dont on veut suivre l'évolution. Tous les objets temporels sont définis comme des sous-classes de `ObjetTemporel`. Des méthodes définissent les liens entre entités ou des opérations temporelles de base.

```
Objet subclass: #ObjetTemporel
	instanceVariableNames: 'nature valeur marque'
```

```
ObjetTemporel subclass: #Etat
	instanceVariableNames: 'fin validite'
```

```
ObjetTemporel subclass: #Evenement
```

Un événement est référencé sur un axe temporel. La granularité temporelle dépend de la finesse nécessaire à l'application. Les relations d'*antériorité* (<), relation d'ordre linéaire et antisymétrique (irréversibilité du temps), et de *simultanéité* (=) sur les dates, permettent de définir les relations correspondantes sur les événements. A l'aide d'opérations arithmétiques sur les dates, nous pouvons calculer des distances temporelles entre événements, générer des événements décalés (avancés ou reportés) dans le temps, ...

Les états sont associés à des intervalles de temps. Les comparaisons entre intervalles sont basées sur la comparaison des dates associées à leurs bornes supérieures et inférieures. Une relation d'antériorité entre états est ainsi disponible. Toutes les relations de Allen sont accessibles sous forme de méthodes attachées à la classe `Intervalle`. Puisque certains états ne durent pas continûment sur un intervalle, leur durée exacte ne correspond pas à celle obtenue par la différence entre la borne supérieure et la borne inférieure de l'intervalle associé. Nous avons introduit la notion d'intervalle discontinu qui permet, par l'ajout d'une borne intermédiaire variable, de représenter cet aspect[5]. Les opérations d'union (concaténation), d'intersection et de différence entre intervalles sont pareillement définis.

La figure 4 montre l'extension réalisée pour représenter notre expertise à partir des structures temporelles initiales. Les diagnostics sont considérés comme des instances de la classe `Diagnostic` sous-classe de `Événement`. Les diagnostics vont caractériser les états respiratoires du patient (instances de `EtatVentilatoire` (EV) sous-classe de `Etat)` valables sur une période de temps. Les actions thérapeutiques (par exemple une action sur le ventilateur `ActionVent`) sont aussi des instances d'une sous-classe de `Evenement`, la classe `Action`. Les états souhaités ou prévus par l'expert sont regroupés comme sous-classes d'`EtatFutur`.

3.6.2. Les Inférences Temporelles

Nous distinguons deux types de connaissances temporelles : un discours général qui s'applique aux objets de base de notre représentation et un discours spécifique, contenant les connaissances du réanimateur, qui porte sur les objets particuliers à son expertise.

Les relations du premier ordre : *termine*, *initie*, qui indiquent en quoi l'apparition d'un événement modifie les objets temporels existants, et *apparaitA* qui le situe

---

[5] Le patient est ventilé depuis ce matin 10 heures ne signifie pas qu'à 15 heures il a été ventilé 5 heures. Le patient a pu être déconnecté de la machine pour des examens, ..., à diverses reprises. L'intervalle associé à l'état "ventiléArtificiellement" aura une date de départ, une valeur représentant la durée exacte de ventilation jusqu'au dernier débranchement, la date du dernier rebranchement et éventuellement une date de fin. Cette notion d'intervalle discontinu peut être rapprochée de celle de processus [Allen 84].

temporellement sont représentées comme des méthodes de la classe `Evenement`. Elles seront ainsi comprises par tous les objets temporels qui modifient l'état du monde et qui sont, par définition, des sous-classes de `Evenement`.

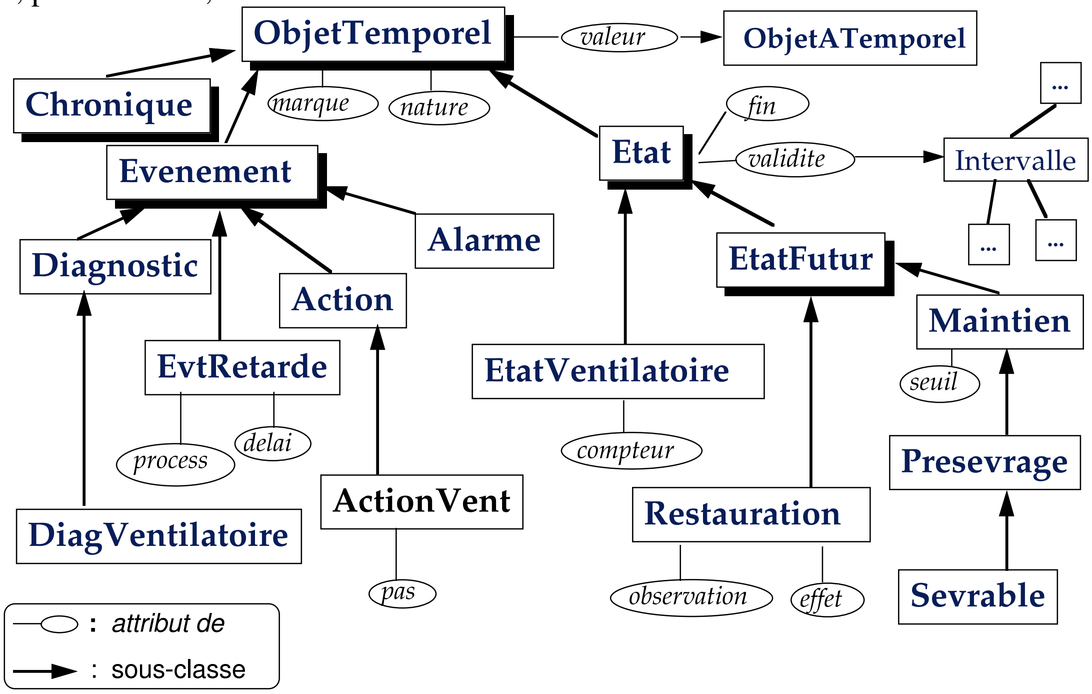

Figure 4. *Les structures temporelles.*
Les trois classes : `Evenement`, `Etat` et `Chronique` représentent les objets temporels de base de notre système. Elles sont indépendantes de l'expertise et sont placées dans des rectangles ombrés. Des sous-classes sont créées en fonction des besoins de représentation. L'apparition certaine d'événement dans le futur est représentée par la classe `EvenementRetarde`. Une métrique temporelle associe une date d'apparition à un objet de type événement ou un intervalle à un objet de type état. Les états futurs regroupent les divers états prévus ou souhaités : maintien de la situation actuelle (`Maintien`), état précédent le sevrage (`Presevrage`), restauration d'un état but (`Restauration`), patient sevrable (`Sevrable`). Plusieurs sous-classes d'événements sont introduites : des diagnostics (`DiagVentilatoire`) et actions (`ActionVent`) spécifiques ainsi que des alarmes (`Alarme`).

```
!Evenement methodsFor: 'definition'!

termine: unObjetTemporel   "Relation r1"
        ^unObjetTemporel fin: self

initie: unObjetTemporel  "Relation r2"
        ^unObjetTemporel marque: self

apparaitA: uneDate  "Relation r3"
        self marque:  uneDate
```

Les connaissances du domaine et les connaissances de contrôle sont représentées séparément. Deux hiérarchies de bases de règles sont nécessaires : à chaque base de connaissances du domaine est associée une base de connaissances de contrôle. Il y a une graduelle spécialisation des connaissances lors de la descente dans l'arbre hiérarchique : les bases de règles du plus bas niveau hiérarchique représentent des connaissances très spécifiques, alors que les connaissances de plus haut niveau sont générales, indépendantes de l'application. Rappelons que les règles (et métarègles) sont triées, par défaut, de façon à privilégier le déclenchement des plus spécifiques.

A partir de la classe abstraite, racine des bases de règles, `NeOpusRuleSet` et de la base de métarègles `DefaultMeta`, qui définit le cycle standard d'évaluation d'une base de règles (envoi des objets et formation de l'espace des conflits, sélection d'une règle et application de cette règle), la figure montre l'organisation des connaissances temporelles.

Pour représenter le raisonnement temporel de l'expertise nous utilisons ou réutilisons 35 classes (17 sont indépendantes du domaine) et, sur trois niveaux hiérarchiques, 20 règles (6 sont réutilisées et 6 sont redéfinies dans les sous-classes), et 18 métarègles (6 sont réutilisées et 8 sont redéfinies dans les sous-classes) (voir figure 5).

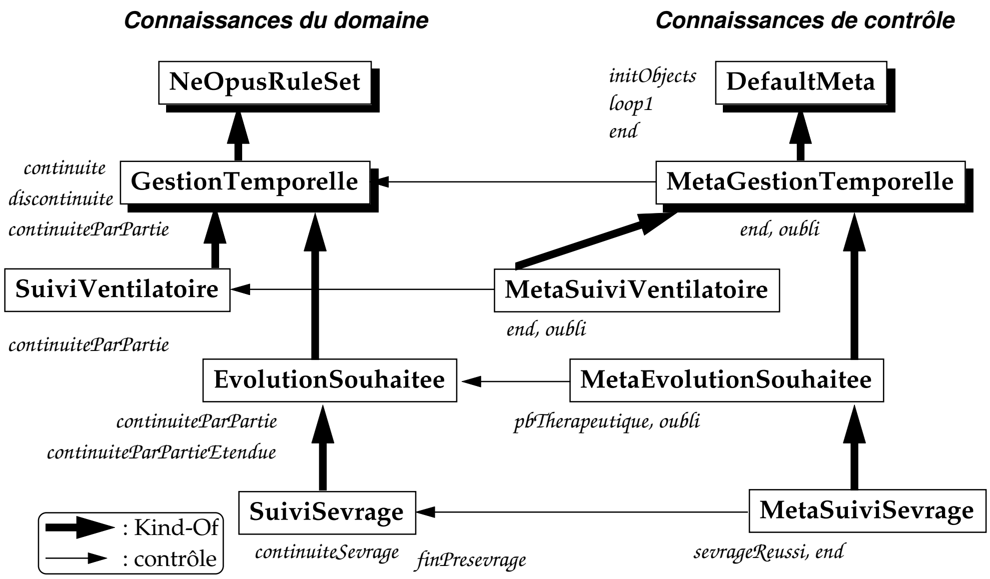

Figure 5.*Organisation des connaissances temporelles.*
Les bases de règles sont organisées hiérarchiquement. Seules les bases de règles non ombrées dépendent de l'expertise. `NeOpusRuleSet` *et* `DefaultMeta` *sont respectivement une base de règles et une base de métarègles standards à l'environnement NéOpus et indépendantes du domaine.* `GestionTemporelle` *et* `MetaGestionTemporelle` *sont ajoutées pour gérer des informations temporelles et sont indépendantes de l'application. Chaque base de niveau méta contrôle le déclenchement des règles de la base à laquelle elle est liée et gère un mécanisme d'oubli d'informations qui lui est propre. Quelques règles pour chaque base de règles (ou de métarègles) sont indiquées sur la figure. Des règles peuvent être ajoutées à celles héritées. Une règle est redéfinie dans une sous-base si elle porte le même nom qu'une règle héritée des super-bases (exemple* `continuiteParPartie` *pour les connaissances du domaine et oubli pour les connaissances de contrôle).*

3.6.3. Représentation de l'Agrégation

Agrégation et oubli sont les deux mécanismes clés pour représenter le processus de raisonnement. Nous avons insisté sur le fait que ces deux mécanismes requièrent des connaissances dépendantes du domaine. Grâce au mécanisme d'héritage des bases de règles présent dans NéOpus, nous introduisons progressivement les nécessaires considérations liées au domaine.

3.6.3.1. Connaissances Générales pour l'Agrégation

Les abstractions temporelles générales sont représentées sous forme de règles de production d'ordre 1 regroupées dans la base de règles `GestionTemporelle`. Chaque prémisse et conclusion d'un axiome apparaissent respectivement en partie condition et en partie action de la règle correspondante (voir figure 6). En partie déclaration, sont filtrées les instances de la classe racine `Etat` et par défaut les instances de ses sous-classes (typage naturel). Les prémisses négatives (préfixe NOT) assurent qu'il n'existe pas d'objet de même nature entre les deux états à comparer.

```
continuite "Axiome 4
|Etat e1 e2|
e1 persistant.
e1 equivalent: e2.
e1 < e2.
NOT | Etat e3 | (e3 ~~ e1 & (e3 ~~ e2)) and: [(e3 memeNatureQue: e1) & (e3 entre: e1 et: e2)].
actions
e2 remove.
e1 incrementeDe: e2 duree.

discontinuite "Axiome 5"
|Etat e1 e2|
e1 persistant.
e1 contredit: e2.
e1  < e2.
NOT | Etat e3 | (e3 ~~ e1 & (e3 ~~ e2)) and: [(e3 memeNatureQue: e1) & (e3 entre: e1 et: e2)].
actions
e2  apparition termine: e1.

continuiteParPartie "Axiome 6"
|Etat e1 e2 e3. Duree d1 |
e3 persistant.
e1 equivalent: e3.
e2  contredit: e3.
e2  entre:e1 et: e3.
e2 duree <= d1 valeur.
actions
e1 validite: (IntervalleDiscontinu with: e1 dateApparition with: e3 dateApparition with: e1 duree).
e2 remove. e3 remove.
e1 incrementeDe: e3 duree.

cesseSurUneDuree "Axiome 7"
|Etat e1. Duree d1|
e1 interrompu.
NOT |Etat e2| (e1 ~~ e2 & (e2 equivalent: e1)) and: [ e2 dateApparition < (e1 fin + d1 valeur)].
actions
Transcript show: e1 valeur, 'apparu à', e1 dateApparition  printString, 'cesse durant : ', d1 valeur printString.
```

Figure 6. *Extrait de la base de connaissances* `GestionTemporelle`.
La règle continuite représente l'axiome 4. Elle se lit de la façon suivante : Quels que soient les objets e1 et e2 instances de la classe `Etat`*, SI e1 est persistant ET e1 est équivalent à e2 ET e1 précède e2 ET qu'il n'existe pas d'objet e3, instance de la classe* `Etat`*, tel que e3 soit de même nature que e1, qu'il suive e1 et précède e2, ALORS éliminer e2 et incrémenter la durée de e1 de la durée de e2. Les autres règles se lisent de la même façon. Le message* `remove` *(règles* `continuite` *et* `continuiteParPartie`*) permet d'éliminer du réseau Rete un objet donc de l'oublier lors des phases futures du raisonnement.*
La conclusion de la règle `continuite` *a pour effet d'incrémenter la durée de l'état e1 et d'éliminer l'information redondante.*
La conclusion de la règle `discontinuite` *a pour effet de terminer l'état en cours.*
La règle de `continuiteParPartie` *associe un nouvel intervalle de type discontinu à l'état e1 ; les deux autres états sont éliminés.*

3.6.3.2. Perception du changement

Les méthodes `equivalentA:` et `contredit:` apparaissent dans les prémisses des règles pour juger si deux objets temporels doivent ou non être considérés comme *similaires ou différents*. Elles sont redéfinies dans chaque sous-classe de `Etat` pour incorporer les informations spécifiques requises pour apprécier le degré de changement. L'interprétation des conditions de déclenchement d'une règle sera modifiée en fonction du type d'objets temporels filtrés.

La méthode `memeNatureQue:` indique si deux objets représentent le même type d'information et peuvent donc être comparés.

```
!ObjetTemporel methodsFor: 'Comparaison'!
memeNatureQue: unObjet
	^self nature = unObjet nature

!EtatVentilatoire methodsFor: 'Comparaison'!
equivalentA: unEtat
	^self memeNatureQue: unEtat & (self valeur = unEtat valeur)

contredit: unEtat
	^self  memeNatureQue: unEtat & (self valeur ~= unEtat valeur)
```

```
!Restauration methodsFor: 'Comparaison'

equivalentA: unEtat
	self class = unEtat class ifFalse: [(self class subclasses includes: unEtat class)
			ifFalse: [^false]].
	^self memeNatureQue: unEtat & (self effet = unEtat effet)

contredit: unEtat
	self class = unEtat class ifFalse: [(self class subclasses includes: unEtat class)
			ifFalse: [^false]].
	^self memeNatureQue: unEtat & (self effet ~= unEtat effet).
```

Par exemple, une instance de la classe `Etat` sera équivalente à (inversement contredira) un état de type `EtatVentilatoire` si elle représente une information de type respiratoire (`memeNatureQue:`) et si leurs valeurs sont les mêmes (inversement sont différentes). Une instance de la classe `Etat` sera équivalente à (inversement contredira) un état de type `Restauration` si elle appartient à la classe `Restauration` ou à ses sous-classes, si leur but est la restauration de la même fonction (`memeNatureQue:`) et leurs effets souhaités sont les mêmes (inversement sont différents).

3.6.3.3. Le Discours Spécifique pour l'Agrégation

Un discours temporel spécifique est modélisé en redéfinissant uniquement certains axiomes et en enrichissant le noyau d'axiomes initial. En utilisant le mécanisme d'HBR de NéOpus, nous avons créé des sous-bases de la base de règles générale `GestionTemporelle` qui héritent des règles d'abstractions temporelles contenues dans cette dernière. Chacune de ces sous-bases représente une phase d'abstraction temporelle spécifique, correspondant à une des étapes du raisonnement.

La sous-base `SuiviVentilatoire` va permettre de gérer la première étape (RT1) du raisonnement temporel. Elle hérite des règles qui gèrent la concaténation d'objets temporels ou la reconnaissance d'états de rupture de la stabilité.

```
GestionTemporelle subbase: #SuiviVentilatoire
```

L'axiome nécessaire pour représenter la connaissance suivante : "nous admettrons que le patient est globalement stable, si l'instabilité a été inférieure à un cycle d'expertise", est introduit par la nouvelle définition de la règle continuiteParPartie héritée de la super-base GestionTemporelle. Cette règle s'applique dès lors à des objets de type EtatVentilatoire et ne filtre plus, comme initialement, des objets généraux de type Etat et de type Duree (voir Figure 7).

```
!SuiviVentilatoire methodsFor: 'agregation'!

continuiteParPartie "Axiome 8"
|EtatVentilatoire e1 e2 e3 |
e3 persistant.
e1 equivalent: e3.
e2 contredit: e3.
e2  entre:e1 et: e3.
e3 dureeEnExpertise >1.
e2 dureeEnExpertise  <= 1.
actions
e1 validite: (IntervalleDiscontinu with: e1 dateApparition with: e3 dateApparition with: e1 duree).
e2 remove. e3 remove.
e1 incrementeDe: e3 duree.
```

Figure 7. *Extrait de la base de connaissances SuiviVentilatoire.*
Cette règle redéfinit les conditions d'agrégation pour des objets de type EtatVentilatoire.

De même, la sous-base EvolutionSouhaitee gère le suivi des états futurs (états souhaités et prévus).

```
GestionTemporelle subbase: #EvolutionSouhaitee
	globalObjects: ''
	category: 'GestionTemps'
```

De la même façon que pour SuiviVentilatoire la règle continuiteParPartie héritée de GestionTemporelle est redéfinie pour s'appliquer aux instances de EtatFutur. Des instabilités peuvent être tolérées, sous certaines conditions, sans remettre en cause les prévisions et la thérapie générale adoptée. Pour en tenir compte, une règle continuiteParPartieEtendue est ajoutée (voir Figure 8).

Dans le cas où le patient n'a besoin que d'un faible niveau d'assistance, les étapes thérapeutiques de présevrage (T2) ou de sevrage (T3) sont envisageables. Des objets, instances des classes Presevrage et Sevrable sont générés lorsque ces étapes sont actives. Un raisonnement temporel spécifique s'applique à ces objets particuliers. Il est représenté par la base de règles SuiviSevrage qui redéfinit et augmente les connaissances contenues dans sa superbase EvolutionSouhaitee par un mécanisme similaire.

```
continuiteParPartieEtendue
|Maintien b1  b2 b3 |
b1 duree > selfAgent seuilStabilite.
b3 persistant.
b3  dureeEnExpertise > 3.
b1 equivalent: b3.
b2 contredit: b3.
b2  entre:b1 et: b3.
b2 dureeEnExpertise  <= selfAgent toleranceInstabilite.
actions
b1 validite: (IntervalleDiscontinu with:b1 dateApparition with: b3 dateApparition marque with: b1 duree).
b1 incrementeDe: b3 duree.
selfBase oublieSauf: b1.
```

Figure 8. *Extrait de la base de connaissances* `EvolutionSouhaitee`.
Des instabilités par rapport à l'évolution souhaitée ne remettent pas en cause la stratégie générale si leur durée est tolérable (`toleranceInstabilite`*) et si elles apparaissent dans un contexte satisfaisant par ailleurs stable (prémisses 1 et 3).*

3.6.4. Représentation de l'Oubli

Les objets persistent dans le réseau Rete tant qu'ils ne sont pas enlevés explicitement [Forgy

1994]. Ainsi coexistent dans ce réseau des objets représentant des situations d'un passé plus ou moins lointain, auxquelles s'ajoutent au cours du temps les situations qui représentent le présent. Le raisonnement modélisé (les inférences) s'applique à tous les objets placés dans le réseau. L'expert en revanche ne raisonne qu'avec un nombre limité de situations qu'il a triées et retenues dans sa mémoire à court terme (de l'ordre de 5 à 9 selon [Miller

1956]). Les autres ont été écartées, oubliées. Puisque des objets s'amoncellent dans l'équivalent artificiel de la mémoire à court terme, - le réseau Rete -, sans participer activement au raisonnement tout en pouvant le perturber, des connaissances vont gérer un mécanisme d'oubli, pour ôter les objets périmés, les transférer dans la mémoire à long terme, - soit la mémoire vive et/ou un disque -, d'où ils pourront éventuellement être rappelés.

L'*oubli actif* est implémenté en utilisant en partie action d'une règle 1) la méthode `remove` (voir règles continuite ou continuiteParPartie) qui a pour effet d'ôter le receveur du réseau Rete ou bien, 2) la méthode `oublieSauf:` (voir, e.g. la règle `continuiteParPartieEtendue`) appliquée à la base de règles pour enlever tous les objets sauf un.

Des règles peuvent être ajoutées pour gérer l'oubli en fonction du contexte. La règle `finPresevrage` (voir Figure 9) indique lorsque la troisième phase thérapeutique (T3) peut être déclenchée (fin de l'observation de présevrage), un état `Sevrable` est déclaré. Dès lors qu'une nouvelle stratégie se met en place, les états précédents sont oubliés.

```
!SuiviSevrage methodsfor: 'sevrage'!

finPresevrage
|PreSevrage p1|
p1 duree > selfAgent seuilSevrage.
p1 sevrable not.
actions
TranscriptUser cr; show: ' Patient Sevrable à :  ', Temps dateEtHeure printString .
selfBase oublieSauf: (Sevrable new).
```

Figure 9. *Extrait de la base de connaissances `SuiviSevrage`.*
Le déclenchement d'une nouvelle phase thérapeutique conduit à l'oubli de certaines informations passées.

L'*oubli passif* est déclenché lorsqu'un certain nombre d'objets sont présents dans le réseau ou bien lorsque des informations ne sont plus utilisées. Pour gérer ce type d'élimination, nous avons introduit des connaissances de contrôle qui, lorsqu'elles sont activées, ne laissent persister que les objets utiles au discours en cours. Nous les avons représentées d'une manière déclarative sous forme de métarègles indépendantes du discours temporel. Elles sont basées sur divers critères et sont totalement redéfinissables.

La métarègle classique de fin de consultation (`end`), définie dans la métabase racine `DefaultMeta`, est réécrite dans sa sous-base `MetaGestionTemporelle.`

```
DefaultMeta subbase: #MetaGestionTemporelle
	globalObjects: ''
	category: 'GestionTemps'
```

Lorsque dans un nœud quelconque du réseau, le nombre d'objets présents dépasse un seuil critique, et si de nouvelles déductions qui modifient l'état des choses n'ont pas été effectuées récemment (l'état courant persiste), le réseau Rete est purgé et seul l'état courant est conservé (`e ruleBase oublieSauf:`). Un objet `EvaluateurSuivi` est créé avec des valeurs de seuils appropriées et redéfinissables (voir Figure 10, la métarègle `oubli`).

```
!MetaGestionTemporelle methodsfor: 'oubli'!

end
|EvaluateurSuivi e|
e status = #end.
e ruleBase nbreTokenMaxAUnNœud  <=e seuilMemoire.
actions
e suspend.

oubli
|EvaluateurSuivi e|
e status = #end.
e ruleBase nbreTokenMaxAUnNœud  > e seuilMemoire.
e  ruleBase result persisteDurant: e seuilDuree.
actions
e ruleBase oublieSauf: e ruleBase result.
e suspend.
```

Figure 10. *Extrait de la base de connaissances `MetaGestionTemporelle.`*
Les conditions de fin de consultation sont modifiées pour ajouter les conditions d'activation du mécanisme d'oubli.

Avec les méthodes suivantes définies dans la base de règles :

```
!GestionTemporelle class methodsFor: 'oubli'!

nbreMaxDeTokensAUnNoeud
	| list max |
	list := self network listOfNodes collect: [:e | e newTokens size].
	max := list first.
	list do: [:elt | elt > max ifTrue: [max := elt]].
	^max

oublieSauf: unObjet
	self flushTokens; acceptObject:unObjet.
```

Si après un certain nombre de consultations de la base de règles, le résultat obtenu se maintient (persistance), une métarègle de la métabase `MetaSuiviVentilatoire,` plus spécifique que la précédente, purge le réseau (voir Figure 11).

```
!MetaSuiviVentilatoire methodsfor: 'oubli'!

oubli
| EvaluateurSuivi e  |
e status = #end.
e nbreConsultation >= e seuilMemoire.
e  ruleBase result persisteDurant: e seuilDuree.
actions
e ruleBase oublieSauf: e ruleBase result.
e suspend.
```

Figure 11. *Extrait de la base de connaissances* `MetaSuiviVentilatoire`*. Redéfinition des conditions d'oubli dans les sous-métabases de règles.*

Les objets `Evaluateur` filtrés par les métabases de règles représentent l'état courant du raisonnement. Ces objets sont aussi raffinés par héritage pour les besoins du contrôle. La figure 12 montre l'arbre d'héritage des objets de contrôle. `EvaluateurMarque` permet de mémoriser le nombre de consultations effectuées pour une base de règles donnée. `EvaluateurSuivi` est utilisé par `MetaGestionTemporelle` et `MetaSuiviVentilatoire` pour gérer l'oubli d'informations.

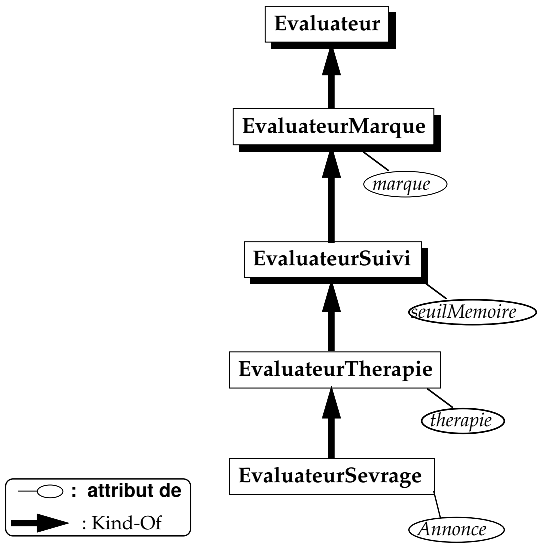

Figure 12. *Un extrait de la hiérarchie d'héritage pour les objets de contrôle.*

3.7. Discussion

Pour construire notre modèle nous avons retenu, à partir d'expertises cliniques, deux mécanismes pour apprécier l'évolution du patient : l'agrégation et l'oubli d'informations. Nous pensons que l'agrégation et l'oubli sont des mécanismes cognitifs inhérents au raisonnement de l'expert. Ils ne sont donc pas introduits uniquement pour améliorer la résolution informatique du problème mais pour mimer le raisonnement de l'expert.

Nous avons modélisé l'évolution du patient comme un problème de transition d'états et le changement comme un processus discret. Les événements introduisent un changement abrupt et instantané ; après l'apparition d'un événement un nouvel état est introduit. Bien que, comme l'a noté Allen, les transitions dans le monde réel prennent du temps, nous considérons que leur assimilation à un point temporel est suffisante pour la plupart des applications. Nous mélangeons une représentation à l'aide de points (événements) et d'intervalles (états). Toute proposition évolutive est encadrée par des événements qui la font naître et s'étendre. Dans un premier temps, nous identifions des événements qui se produisent dans le monde réel et dans un second temps nous identifions les propositions modifiées par l'apparition de ces événements. Lorsqu'un diagnostic (D1) introduit un nouvel état différent du précédent, nous considérons que cet état dure depuis le dernier diagnostic (D0) effectué, bien qu'en réalité la date du changement d'état se situe quelque part entre D0 et D1. Les diagnostics se répétant à une fréquence relativement importante (toutes les 2 minutes), la gestion d'une fenêtre temporelle liée à un événement aurait peu d'intérêt. Nous avons une logique réifiée du premier ordre. Pour représenter une proposition telle que "le patient est tachypnéïque de 7h à 8h", nous introduisons un état ventilatoire de type OT<t,p> avec p = 'est tachypnéïque', auquel sera associé une marque temporelle t, soit un intervalle de validité I[7, 8]. La proposition 'est tachypnéïque' est transformée et traitée comme un objet, terme d'un prédicat, et non comme un prédicat. On aura valide(tachypnéïque (patient), I). Dans les conditions des règles temporelles, c'est la forme prédicative (valide(p,t)) qui est utilisée : chaque condition de règle renvoie un booléen. Cependant si valide(p,t) est vraie, il n'y a pas forcément un objet temporel de type OT<t,p> qui le traduit. L'information contenue dans valide(p,t) peut être distribuée entre plusieurs OT (avant agrégation). Elle peut aussi être englobée dans un OT notamment celui qui s'est étendu par l'opération d'agrégation à l'objet qui traduisait exactement valide(p,t).

Notre approche présuppose que nos connaissances temporelles sont certaines. De plus, nous supposons que les actions ne peuvent pas engendrer de changements graduels continus. Nous considérons que ces aspects ne sont pas cruciaux pour les applications visées. Des considérations similaires ont été faites dans d'autres contextes médicaux (suivi de l'évolution hémodynamique [Lau and Vincent 1993], suivi de l'évolution des taux de cholestérol, [Rucker et al. 1990]).

Notre réflexion est proche de celle de [Shahar and Musen 1993; Shahar 1994] avec le récent système RESUME qui effectue des abstractions sur des données temporelles pour le monitorage de patients.

Leur système est appliqué aux thérapies pour des maladies chroniques (HIV par exemple) dont la gestion est plus complexe que des situations où une thérapie peut être administrée en temps réel (comme la ventilation artificielle). En effet, des événements peuvent à tout instant remettre en question des abstractions précédemment produites. Cette révision est implémentée par un module de maintenance de vérité (TMS). Par ailleurs, RESUME utilise des fonctions spécifiques et des tables d'abstractions pour chaque paramètre (état, taux ou gradient) pour apprécier les degrés de changement. Pour apprécier l'évolution nous n'utilisons que de simples comparaisons entre états.

4. Un modèle du temps pour l'analyse harmonique automatique

Le deuxième système que nous présentons ici traite d'un problème - en apparence - bien moins critique : l'analyse musicale. Nous verrons plus loin que les deux problèmes entretiennent des relations très étroites, par le biais de leur modélisation.

4.1. Position du problème

Le problème de l'analyse harmonique automatique consiste à produire pour un corpus musical donné, une analyse harmonique de l'ensemble de ses constituants. Dans le cas des séquences d'accords, il s'agit plus précisément d'identifier les tonalités sous-jacentes pour chacun des accords de la grille. La particularité de ce problème est que l'analyse à produire est hiérarchique. Ainsi, on peut dire qu'un morceau dans son ensemble est en do majeur, alors que certaines parties sont dans une autre tonalité (modulation). De même, pour une sous partie donnée, des modulations locales sont possibles. De manière générale, l'analyse harmonique consiste donc à produire un arbre d'analyse permettant d'interpréter chaque accord de la séquence à plusieurs niveaux (Cf. Figure 15, le résultat de l'analyse de Blues For Alice Figure 14).

Enfin, le résultat de l'analyse est de donner pour chaque accord la tonalité sous-jacente, à partir de laquelle le musicien pourra alors improviser, arranger ou accompagner.

4.1.1. La théorie harmonique

L'harmonie tonale est un domaine relativement bien formalisé. Cet effort de formalisation existe depuis plusieurs siècles, et on trouve aujourd'hui des traités d'harmonie complets exposant la théorie sous-jacente à l'agencement de notes en musique tonale dans divers styles [Bitsch 1957; Coker 1964; Beaudoin 1990]. Cependant, la situation en ce qui concerne l'analyse est comparable à ce qui se passe pour les langues naturelles: si de nombreux travaux ont été faits pour donner à la langue une structure formelle, une syntaxe, voire une grammaire, il n'existe pas de théorie computationnelle complète pour l'analyse. En revanche, le problème de l'analyse automatique est bien sûr moins compliqué que celui de l'analyse des langues naturelles, car c'est un problème bien plus restreint.

La théorie harmonique pour les séquences d'accords peut se décrire en deux points principaux. D'une part un principe de "légalité" postule que chaque accord, *pris isolément,* peut être analysé, *in abstracto*, dans un certains nombre de tonalités. Ces tonalités sont les gammes (ensembles de notes) qui contiennent toutes les notes de l'accord. Par exemple, l'accord de C majeur, tout seul, peut être analysé comme: 1^{er} degré de la gamme Do majeur, 4^{e} degré de la gamme de Sol majeur, 5^{e} de Fa majeur, 6^{e} de Mi mineur harmonique, etc. L'ensemble de ces tonalités légales est le résultat d'un calcul totalement déterministe [Pachet 1994b].

D'autre part, le choix de la "bonne" tonalité va ensuite se faire en fonction du contexte de l'accord, c'est à dire de ses accords voisins, et ce bien sûr, récursivement. La prise en compte du contexte suit alors un principe de minimalité: on va essayer de trouver une analyse du morceau qui minimise le nombre de modulations, c'est à dire que l'on va chercher les tonalités qui sont communes au plus grand nombre d'accords adjacents possibles.

L'analyse se complique par ailleurs par plusieurs phénomènes qui échappent à la formalisation rigoureuse, mais qui sont néanmoins essentiels.

D'une part la présence d'accords de substitutions obscurcit le principe de légalité. Par exemple, un accord de septième de dominante peut être remplacé par un accord de septième sur le triton (G7 peut être remplacé par Db 7, etc.). Par ailleurs des formes particulières sont analysables directement. Ainsi, un "2-5-1" est-il reconnaissable directement, et analysable dans une tonalité fixe. Par exemple, la séquence "D min 7 / G 7 / C maj 7" est un "2-5-1" analysable en C majeur. De même, la séquence "C maj 7 / Eb 7 / Ab maj 7 / Db maj 7" est une anatole substituée, dont un musicien reconnaîtra immédiatement la nature (en do majeur aussi). Notons que ces structures, commes les accords de substitution, ne suivent pas toujours les règles de cohérence harmonique.

Ainsi, dans l'exemple précédent, l'accord de Db maj 7 n'est pas, *in abstracto*, analysable en C majeur. Ce n'est qu'au sein d'une telle séquence qu'il peut le devenir.

Enfin, le processus d'analyse est hiérarchique. Ceci veut dire qu'une structure reconnue (un "2-5", une anatole, un turnaround, etc) peut elle-même être considérée comme atomique, et incorporée dans une structure de plus haut niveau. On peut ainsi considérer des accords de septième de dominante comme des préparations d'accords, et ainsi analyser des cascades de résolutions. Par exemple, la séquence: "A 7 / D 7 / G 7 / C" peut être analysée entièrement en Do majeur, par le processus hiérarchique illustré par la figure 13).

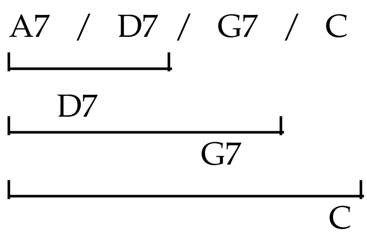

Figure 13. *Une vision hiérarchique de l'analyse d'un groupe d'accords.*

Enfin, a un niveau plus haut, apparaissent les structures globales d'un morceau. Ainsi un blues est identifié par une séquence de 12 temps, ayant trois sous-séquences couvrantes, et telles que la première et la troisième sont dans la même tonalité, et la deuxième est dans la tonalité de la première transposée d'une quarte. De même pour les structures AABA, AABB, etc.

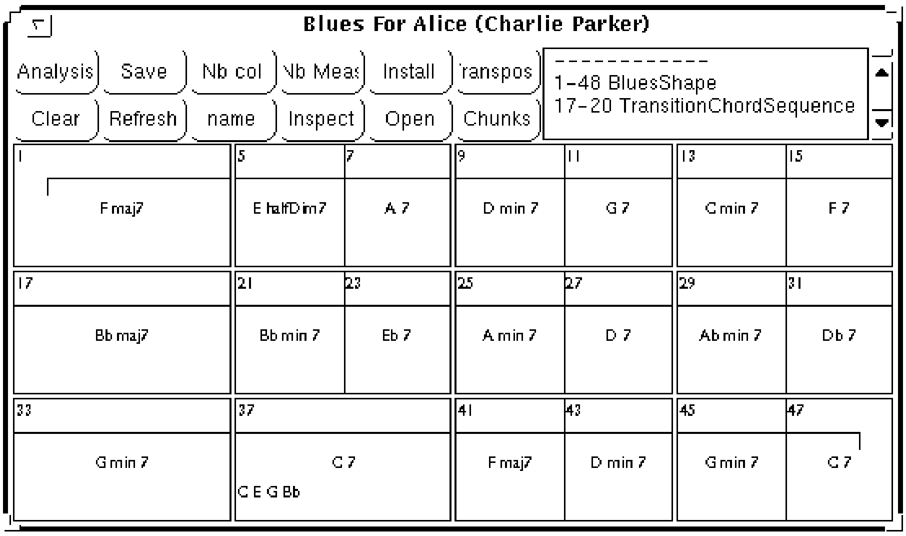

Figure 14. *Une séquence d'accords modélisée en MusES.*

Analyse de Blues ForAlice

1-48 BluesShape in F MajorScale
  45-4 ResolvanteEndToBeginning in F MajorScale
    45-48 ChordSubstitution in F MajorScale
      45-48 TwoFive in F MajorScale
        [G #min 7]
        [C 7]
    [F #maj7]
  5-20 ChordSubstitution in Bb MajorScale
    5-20 Resolvante in Bb MajorScale
      5-16 ChordSubstitution in Bb MajorScale
        5-16 TwoFive in Bb MajorScale

```
        5-14 ChordSubstitution in C HarmonicMinor
          5-14 Resolvante in C HarmonicMinor
            5-12 ChordSubstitution in C MajorScale
              5-12 TwoFive in C MajorScale
                5-10 ChordSubstitution in D HarmonicMinor
                  5-10 Resolvante in D HarmonicMinor
                    5-8 ChordSubstitution in D HarmonicMinor
                      5-8 TwoFive in D HarmonicMinor
                        [E #halfDim7 ]
                        [A 7 ]
                    [D #min 7 ]
                [G 7 ]
            [C #min 7 ]
        [F 7 ]
    [Bb #maj7 ]
17-48 ExtendedShape in F MajorScale
  17-46 ExtendedShape in F MajorScale
    17-44 ExtendedShape in F MajorScale
      [Bb #maj7 ]
      21-44 ExtendedShape in F MajorScale
        21-42 ChordSubstitution in F MajorScale
          21-42 Resolvante in F MajorScale
            21-40 ChordSubstitution in C MajorScale
              21-40 Resolvante in C MajorScale
                21-32 ChordSubstitution in Db MajorScale
                  21-32 Resolvante in Db MajorScale
                    21-28 ChordSubstitution in D MajorScale
                      21-28 Resolvante in D MajorScale
                        21-24 ChordSubstitution in Ab MajorScale
                          21-24 TwoFive in Ab MajorScale
                            [Bb #min 7 ]
                            [Eb 7 ]
                        25-28 ChordSubstitution in G MajorScale
                          25-28 TwoFive in G MajorScale
                            [A #min 7 ]
                            [D 7 ]
                    29-32 ChordSubstitution in Gb MajorScale
                      29-32 TwoFive in Gb MajorScale
                        [Ab #min 7 ]
                        [Db 7 ]
                33-40 ChordSubstitution in F MajorScale
                  33-40 TwoFive in F MajorScale
                    [G #min 7 ]
                    [C 7 ]
            [F #maj7 ]
        [D #min 7 ]
    [G #min 7 ]
  [C 7 ]
```

Figure15. *Le résultat de l'analyse de la grille de la figure 14.*

4.2. État de l'art en analyse automatique

L'analyse harmonique automatique a fait l'objet de nombreux travaux en intelligence artificielle. Historiquement les premières tentatives ont porté sur l'utilisation de *grammaires musicales*, inspirées des travaux de Chomsky, et revisités par

[Lerdahl and Jackendoff 1983]. Si les grammaires semblent à première vue particulièrement bien adaptées pour représenter les structures récursives inhérentes à la musique tonale, leur utilisation pour produire de systèmes effectifs d'analyses, complets, et entièrement automatiques s'est vite avérée problématique. Plusieurs extensions de formalismes de grammaires ont été proposés (en particulier les grammaires systémiques, [Winograd 1993], voir [Roads

1988] pour une revue complète des formalismes de grammaires et leur utilisation musicale), pour analyser des corpus musicaux dans un style donné (Bach, Mozart), mais cette tendance a été peu à peu abandonnée, devant les problèmes de tractabilité et de fabrication d'analyseurs automatiques (Cf. [Deliège

1994]). Un des résultats les plus intéressants de ces travaux est l'identification, pour des corpus restreints, de grammaires qui permettent de décrire, sur le papier, le corpus donné de manière compacte, sans proposer de mécanisme d'analyse correspondant (Cf. par exemple les règles de [Steedman

1984] pour les grilles de blues, ou [Baroni 1984] pour des grammaires décrivant les corpus de chants esquimaux). Cette opinion peut être agrémenté de justifications cognitives, comme le fait [Laske 1993] qui interprète les grammaires comme des "explications de compétence musicale".

D'autres approches ont été abordées pour résoudre ce problème, comme les approches purement procédurales [Ulrich 1977], par règles de production [Maxwell

1992], ou encore par contraintes et multi-agents [Steels 1979]. Chaque approche a su résoudre un sous-problème du problème d'analyse, mais aucun consensus ne s'est imposé.

A notre avis, le problème principal dans ces systèmes est de trouver une représentation des contraintes temporelles entre objets musicaux qui soit effectivement bien adaptée au processus d'analyse. Dans les modèles à base de grammaire, le temps n'est représenté qu'implicitement, par la juxtaposition des lexèmes. Si cette représentation sommaire suffit effectivement pour l'utilisation de la grammaire à des fins de génération [Johnson-Laird 1991], elle est largement insuffisante lorsqu'il s'agit d'analyser automatiquement, car de nombreuses connaissances temporelles *spécifiques au domaine* sont alors inexprimées.

4.3. Le modèle de représentation temporelle

Le modèle que nous proposons repose sur une représentation riche des objets et contraintes temporelles, associée à un mécanisme de raisonnement calqué sur celui de l'expert. Le raisonnement est vu comme un processus de *reconnaissance* par lequel des objets de plus en plus abstraits sont construits à partir de configurations significatives d'objets de base. Le processus de reconnaissance est par ailleurs *constructif* car il a pour but de produire l'arbre d'analyse. La reconnaissance d'une forme musicale significative est alors représentée par la construction d'un bout de l'arbre d'analyse.

Nous allons détailler la structure des objets utilisés pour représenter ce raisonnement, puis les règles effectives utilisées.

4.4. Les objets utilisés

Les objets manipulés et nécessaires à la résolution de ce problème sont de nature diverse. Nous les avons classé en trois catégories : les objets relevant de connaissances de base sur l'harmonie, les objets temporels de base, et les objets d'analyse.

4.4.1. Connaissances de base

D'abord la modélisation de la séquence d'accords proprement dite nécessite des connaissances de solfège et d'harmonie élémentaire pour représenter les notions d'accord, de note et d'intervalle, ainsi que les objets temporels, ici la séquence d'accords. Ces objets sont définis de manière générale dans une bibliothèque de classes d'objets musicaux à usage général, le système MusES [Pachet 1994b; Pachet 1994a]. Nous ne détaillerons pas ici les propriétés de ces objets qui ne présentent pas de particularité remarquable dans le cadre de cet article. MusES introduit un certain nombre de classes comme `PitchClass`, `OctaveDependentNote`, `Chord`, `MusicalInterval`, `Scale`, etc., ainsi que les calculs y afférent (transposition, calculs sur les noms, enharmonie, accords engendrés, etc.). Notons que MusES a été conçu pour être indépendante du domaine (bien sûr en restant dans le domaine de la musique tonale). L'intérêt de bénéficier d'une bibliothèque réellement utilisable ici est déterminant, les opérations sur les structures musicales étant loin d'être triviales; le fait de pouvoir les utiliser tels quels est en soi un gain énorme de productivité. Le système contient environ 80 classes et 1200 méthodes. Outre le système d'analyse présenté ici, le système MusES est effectivement utilisé dans plusieurs autres applications de taille substantielle; pour un harmoniseur automatique [Pachet and Roy 1995], un simulateur d'improvisations

[Ramalho and Ganascia 1994; Ramalho and Pachet 1994], et un système tutoriel en cours de réalisation.

4.4.2. Représentation des objets temporels en MusES

Pour ce qui est des objets temporels, MusES propose une représentation par un mécanisme d'héritage et de réinterprétation dynamique de messages. Ce mécanisme est le suivant. Une classe racine `TemporalObject` définit un attribut `lapse`, lui-même instance d'une classe `Lapse` définie par un temps de début et une durée. Les objets temporels significatifs sont définis comme des sous-classes de la classe `TemporalObject`, et héritent donc de toutes les propriétés de cette classe, à savoir essentiellement les calculs d'intersection sur les durées.

Noter que tous les objets musicaux n'ont pas forcément d'équivalent temporel, et, inversement, tous les objets temporels n'ont pas forcément d'équivalent atemporel. Par exemple, la notion de `MusicalSilence` n'a pas de sens dans un contexte atemporel. Inversement, les accords peuvent être soit atemporels, soit temporels, de même pour les notes. Pour les pitch-classes, enfin, elles n'ont de sens que dans un contexte atemporel.

Le problème survient à cause de l'héritage simple. En effet, certaines classes ont ainsi deux représentations, l'une temporelle, l'autre atemporelle. Ainsi, on a par exemple la classe `OctaveDependentNote`, qui représente les notes avec une octave, mais de façon atemporelle. Son équivalent temporel est `PlayableNote`[6], qui elle, est sous-classe de `TemporalObject`. De même pour les couples `Chord` et `PlayableChord` (Cf. Figure 16).

4.4.3. Un mécanisme de délégation automatique

Or, à cause de l'héritage simple, les objets temporels ne peuvent être sous-classes que de `TemporalObject`. Ainsi, de nombreuses propriétés des objets temporalisés ne peuvent être hérités dans la version temporelle. Nous avons proposé un mécanisme très simple permettant de remédier à ce problème sans mettre en cause la simplicité de la conception. Ce mécanisme est un mécanisme de délégation automatique, avec compilation dynamique de méthodes de délégation.

Par exemple, la classe TemporalNote contient une méthode `doesNotUnderstand:` qui compile dynamiquement une méthode de délégation à son objet atemporel `Note`. Cette méthode se décrit de la manière suivante:

Dans la classe `TemporalNote`:

```
doesNotUnderstand: aMessage
    (OctaveDependentNote includesSelector: aMessage selector)
        ifTrue:
            [self compileDelegateToNote: aMessage.
            ^self perform: aMessage selector withArguments: aMessage arguments].
    ^super doesNotUnderstand: aMessage
```

La méthode `compileDelegateToNote:` consiste simplement à fabriquer le texte de la méthode de délégation, puis à le compiler dans la classe `TemporalNote` :

```
compileDelegateToNote: aMessage
   "Fabrication du texte de la méthode de délégation"
   ...
   " et compilation dans la classe"
   self class compile: aStream contents classified: 'auto compiled methods'!
```

Ainsi, la première fois qu'une instance de `TemporalNote` reçoit le message `fourth` par exemple, la méthode suivante est créée:

```
fourth
    ^note fourth
```

Puis le message est renvoyé à l'instance en question, qui délègue, avec retour du résultat. Ce mécanisme est mis en oeuvre pour chaque sous-classe de `TemporalObject`, avec à chaque fois délégation explicite à l'objet atemporel correspondant.

Noter que nous ne prétendons pas résoudre ici le problème de l'héritage multiple, ni celui de la délégation automatique dans sa généralité. Nous proposons une solution simple et efficace qui permet de construire notre ontologie temporelle, et dont le coût est

---

[6] Cette classe pourrait s'appeller TemporalNote, par symétrie. Nous avons préféré le nom PlayableNote, car la classe en questiuon contient aussi un certain nombre de paramètres spécifiques aux notes effectivement entendues (amplitude, canal MIDI, etc.)

négligeable puisque le mécanisme de réification des messages et de compilation n'est effectué qu'une seule fois.

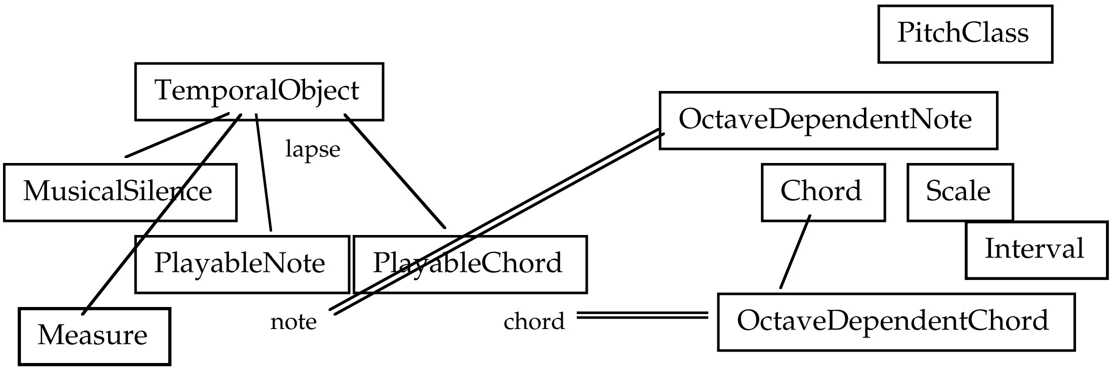

Figure 16. *La hiérarchie d'objets temporels musicaux, et quelques équivalents atemporels.*

4.4.4. Les objets d'analyse

Enfin, le raisonnement d'analyse harmonique fait intervenir des objets plus abstraits, qui vont constituer le résultat de l'analyse à proprement parler. Ces objets sont les formes - invisibles - représentant des groupes d'accords ayant une fonction harmonique précise. On trouve dans cette catégories les structures mentionnées plus haut: "2-5"; anatoles, turnaround, structure de Blues, AABA, ainsi que résolvantes et substitutions.

Nous avons proposé dans [Pachet 1994d] une ébauche de modèle du raisonnement dans les langages à objets + règles, basé sur la distinction entre objets perçus et objets conçus. Dans cette terminologie, les objets d'analyse sont les objets conçus au cours et par le raisonnement, à partir d'observation de configurations d'objets perçus, ici les accords.

Ces objets d'analyse sont des objets temporels, et sont donc représentés par des sous-classes de `TemporalObject`. Par ailleurs, une relation hiérarchique de *composition* est introduite pour ces objets, représentée simplement par un attribut (`subShapes`). La figure 17 montre un extrait de cette hiérarchie.

```
TemporalObject
    ObjectInASequence ('chordSequence' 'conceived')
        AnalysableObject ('analysisList')
            ChordInASequence ('chord' 'before' 'after')
            Shape ('subShapes')
                ChordSubstitution ('chord')
                MusicalShape ('tonality')
                    AmorphousShape ()
                    ComposedShape ()
                        ExtendedShape ()
                            ExtendedIntersectedShape ()
                            GlobalShape ()
                                BiTonalShape ()
                                MonoTonalShape ()
                                QuadriTonalShape ()
                                TriTonalShape ()
                    EmpruntModal ()
                    KnownShape ()
                        BluesShape ()
                        BluesySubstitution ()
                        Resolvante ()
                        TurnAround ()
                        TwoFive ()
                        TwoFiveOne ()
                TransitionChordSequence ('beforeTonality' 'afterTonality')
```

Figure 17. La hiérarchie d'objets d'analyse, tous sous-classe de la racine des objets temporels. Les attributs sont entre parenthèses.

4.5. Les règles et le processus de raisonnement

Le processus de raisonnement est représenté par un enchaînement de bases de règles ayant deux fonctions distinctes. D'une part des règles permettent de *reconnaître* des formes harmoniques à partir de configurations d'accords dans la séquence, ou de formes déjà identifiées. Cette reconnaissances de forme permet de créer des objets plus abstraits à partir d'objets moins abstraits. D'autre part des règles représentent un mécanisme de *destruction* de formes, nécessaire pour éviter l'explosion d'objets. Ces règles représentent le principe de minimalité, et expriment des connaissances sur la redondance de formes entre elles. Nous allons décrire plus en détail chacun de ces types de règles.

4.5.1. Les règles d'agrégation

Les règles d'agrégation suivent toutes le pattern suivant :

> Soient `a1` `a2` instances de (une sous-classe de) `AnalysableObject`
>
> 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 `Shape` couvrant la durée des deux objets agrégés.
> **Etablir** le lien de composition entre le nouvel objet et les deux anciens.

La figure 18 montre deux règles extraites d'une des bases de règles d'agrégation .

```
majorTwoFive
"ex : Dmin7 G7 ..
First chord should NOT have a flat Fifth, so that the whole thing is in major"
   | AnalysableObject c1 c2 . ChordSequence cs|
        cs exists.
        c1 isMinor.
        (c1 hasA: #flatFifth) not.
        c2 isAfter: c1.
        c2 isMajor.
        c2 hasA: #minorSeventh.
        c2 root pitchEqual: c1 root fourth.
actions
        |x s|
        x := TwoFive new.
        x beginBeat: c1 beginBeat; endBeat: c2 endBeat.
        x tonality: (c2 root fourth majorScale); subShapes: (Array with: c1  with: c2 ).
        s := ChordSubstitution new chord: c2; beginBeat: c1 beginBeat; endBeat: c2 endBeat.
        s shape: x. s go.
        cs addAnalysis: s.
```

```
resolvanteMajeur
"ex : A7 D 7 ..."
   | AnalysableObject c1 c2 . ChordSequence cs|
        cs exists.
        c1 isMajor.
        c1 hasA: #minorSeventh.
        c2 isAfter: c1.
        c2 isMajor.
        c2 root pitchEqual: c1 root fourth.
actions
        |x s|
        x := Resolvante new.
        x beginBeat: c1 beginBeat; endBeat: c2 endBeat.
        x tonality: (c2 root majorScale); subShapes: (Array with: c1  with: c2 ).
        s := ChordSubstitution new chord: c2; beginBeat: c1 beginBeat; endBeat: c2 endBeat.
        s shape: x. s go.
        cs addAnalysis: s.
```

Figure 18. *Deux règles de la base de règles de reconnaissances de formes.*

On trouve ausi des règles plus abstraites comme la suivante, qui décrit le phénomène d'*emprunt modale*. Cette règle représente le fait 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 particulières (elle peut être analysée dans la tonalité mineure de la tonalité des deux formes qui l'entoure) (Cf. figures 19 et 20)

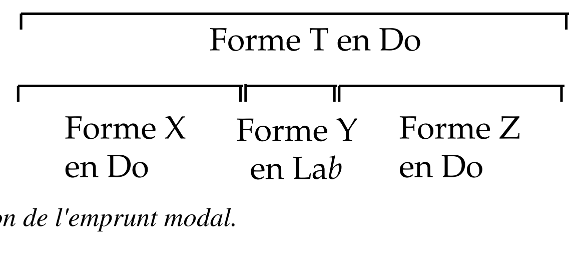

Figure 19. *La configuration de l'emprunt modal.*

```
ShapeExtensionSteedman methodsFor: 'extensions'!
empruntModal
   | Shape c1 c2. ChordInASequence c3. ChordSequence cs|
        cs exists.
        c1 endBeat = (c3 beginBeat - 1).
        c2 beginBeat = (c3 endBeat + 1).
        c1 tonality notNil.
        c2 tonality notNil.
        c1 tonality = c2 tonality.
        c3 possibleScales includes: c1 tonality root minorScale.
actions
        |m e|
        e := EmpruntModal new beginBeat: c3 beginBeat; endBeat: c3 endBeat.
        e subShapes: (Array with: c3).
        e tonality: c1 tonality root minorScale.
        m := ComposedShape new beginBeat: c1 beginBeat; endBeat: (c2 endBeat).
        m subShapes: (Array with: c1 with: e with: c2).
        m tonality: c1 tonality.
```

Figure 20. *La règle de reconnaissance d'une emprunte modal (due à Geber Ramalho).*

4.5.2. Les règles de destruction

Ces règles permettent de supprimer de l'espace de recherche (la mémoire à court terme, représentée par l'état d'instanciation du réseau RETE) des objets devenus inutiles, et qui freinent et perturbent les inférences.

On trouve par exemple des règles comme la règle `removeSubsumedShapes` (Figure 21), qui permettent de supprimer des formes lorsqu'elles sont subsumées par des formes plus grandes, et de même tonalité. Leur suppression n'occasionne alors pas de perte d'information.

```
ShapeSimplificationSteedman2 methodsFor: 'simplifications'!

removeSubsumedShapes
   | ObjectInASequence c1  c2. ChordSequence cs|
        cs exists.
        c1 subsumes: c2.
        c1 ~= c2.
        c2 tonality notNil.
        c1 tonality notNil.
        c1 tonality  = c2 tonality.
actions
        cs removeAnalysis: c2.
        c2 remove.
        c1 modified.
```

Figure 21. *Une règle de destruction de forme musicale.*

4.5.3. L'enchaînement des règles

Le processus de résolution dans son ensemble est décrit par une succession d'exécutions de bases de règles, alternant reconnaissance et simplification. A chaque phase, les bases de règles filtrent des objets d'analyse de plus en plus abstraits. Les différentes bases de reconnaissances représentent différentes connaissances portant sur les structures en fonction de leur complexité. Voici l'ordre de ces phases:

1 - Reconnaissance de formes de base

Règles de substitutions, de formes classiques

Simplification des formes obtenues.

2 - Fabrication de formes particulières pour les accords isolés
Un accord isolé correspond une forme particulière. Cette phase est nécessaire pour pouvoir ensuite traiter uniformément les formes classiques et les accords isolés

Suppression de formes redondantes, dues à des reconnaissances multiples

3 - Grossissement des formes, en fonction de règles d'agrégation générales
Les formes sont grossies à droite et à gauche pour tenter d'inclure des formes adjacentes

Simplification des formes obtenues.

4 - Agrégation de structures
des structures adjacentes sont agrégées quand c'est possible. Des règles particulières sont alors appliquées, comme celle de l'emprunt modal (règle `continuiteParPartie`, Figure 6 d,u système medical, à comparer avec la règle `empruntModal`, Figure 20, dans le système musical).

Simplification des formes obtenues.

5 - Reconnaissance de formes globales
On reconnaît ici les structures comme AABA, Blues, etc.

Simplifications finales et visualisation du résultat sous forme arborescente.

4.6. Discussion

Le système d'analyse fonctionne simplement à l'aide d'un éditeur de grilles (Cf. Figure 14 pour un exemple), qui reproduit l'apparence des grilles telles qu'elles sont dessinées dans les corpus de référence [Real 1981;

Fake 1983]. Un bouton permet alors de lancer le système de raisonnement, pour produire, *in fine*, l'arbre d'analyse (Cf. Figure 15). Cet arbre d'analyse permet ensuite de donner l'analyse de chaque accord, et donc d'en déduire la tonalité sous-jacente. Le système réalisé est capable d'analyser correctement des séquences qui ne passent pas les tests rigoureux des approches à base de grammaire. En particulier, les grilles réputées "difficiles" dans [Steedman

1984] sont analysées sans difficulté. D'autres grilles, qui ne passent pas les tests de la grammaire de [Steedman 1984], sont par ailleurs analysées correctement, en tout ou partie. Une étude complète des corpus de référence est en cours en collaboration avec l'IRCAM, pour affiner le modèle et élaborer une validation complète. Dans sa version actuelle, il est utilisé en entrée du simulateur d'improvisation développé par G. Ramalho [Ramalho and Ganascia 1994; Ramalho and Pachet 1994].

Le système permet d'analyser des séquences de manière partielle. En particulier, une caractéristique importante de ce système est sa résistance aux bruits, i.e. aux séquences d'accords à la limite de la musique tonale. De plus, le processus de raisonnement a une qualité organique intuitive, et se prête donc bien à l'explication.

5. Discussion

La présentation détaillée des deux systèmes à base de connaissances effectuant un raisonnement temporel nous permet maintenant de mettre en évidence des points communs et des points de divergence.

5.1. Points communs

5.1.1. Sur la nature du problème traité

5.1.1.1. Un raisonnement temporel basé sur la représentation de la perception des changements

La perception des changements est dépendante de l'application et ne peut être représentée en général. Pour le système de contrôle de la ventilation, la perception du changement est basée sur des seuils de durée, eux-mêmes redéfinis en fonction du contexte (héritage de classe). Pour le système d'analyse, la perception du changement est représentée par les propriétés harmoniques des accords et formes musicales (leur tonalité principalement).

5.1.1.2. Un raisonnement temporel *hiérarchique*

Le raisonnement sur le temps est dans les deux effectué à plusieurs niveaux d'abstraction, de manière à modéliser l'aspect contextuel de la perception du changement. A ce titre, "apprécier la stabilité" est à rapprocher de "minimiser les modulation". Plus précisément: "Le morceau est globalement un blues en Do (même si il module et contient une majorité d'accords en Fa)" est à rapprocher de "le patient est globalement stable depuis 10 heures (même s'il y a eu divers épisodes courts de ventilation insuffisante)".

5.1.2. Sur le modèle et sa représentation

5.1.2.1. Un même constat, des besoins similaires

Les deux approches ont été développées suite à un constat d'inutilisabilité des modèles formels existants. Dans un cas on refuse les logiques temporelles basées sur les propagations de contraintes dans des graphes, dans l'autre on constate l'inadéquation des modèles à base de grammaire. Cet héritage, curieusement similaire, a des raisons profondes: les modèles généraux sont peu ajustables au domaine d'étude. Dans les deux cas les représentations temporelles choisies exploitent la structure et des connaissances qui dépendent du domaine.

5.1.2.2. Une approche constructive

Les deux approches relèvent d'une modélisation du monde par simulation, et donc d'une vision constructive du raisonnement. Si la modélisation par simulation est particulièrement bien adaptée à la représentation de phénomènes physiques, tangibles et visibles, son utilisation pour représenter des phénomènes de raisonnement (par essence invisibles et singulièrement abstraits) est pour nous un postulat de première importance, qui est imposé par le choix d'un langage de PPO.

5.1.2.3. Une ontologie temporelle simple

Les deux ontologies temporelles sont basées sur un modèle d'objets temporels simple, utilisable par héritage de classe, en conjonction avec le mécanisme de typage naturel de NéOpus. Le mécanisme de délégation automatique est lui spécifique à la technologie Smalltalk. Si les deux ontologies différent dans leurs détails, il y a tout de même des analogies évidentes à faire entre les deux domaines:

Evénement (ventilatoire) correspond à Modulation (musicale)
État (ventilatoire) correspond à Forme musicale
Chronique (ventilatoire) à mélodie et séquence d'accords

5.1.2.4. Deux types de règles : agrégation et oubli

Les deux systèmes mettent clairement en évidence la suffisance de deux types de règles: agrégation et oubli (reconnaissance et destruction). Ceci peut être par ailleurs vu comme un "pattern" récurrent de NéOpus, correspondant à deux opérations conceptuelles bien représentées par la combinaison règle et objet, l'héritage et le typage naturel.

Plusieurs types d'agrégation sont comparables d'un système à l'autre. Par exemple, la règle d'agrégation `continuiteParPartie` est similaire à la règle `empruntModale`.

D'autres systèmes proposent des mécanismes similaires à l'agrégation pour gérer la rapide prolifération d'événements temporels au cours du temps. Par exemple, [Kohane 1987] propose de générer automatiquement ces regroupements sur la base des relations causales entre deux événements ;

[Ash and Hayes-Roth 1990] proposent la concaténation des intervalles sous forme d'épisodes eux-mêmes regroupés à l'aide de critères sémantiques. L'axiome que nous introduisons pour agréger des objets temporels considère que ceux-ci possèdent intrinsèquement la propriété de concaténation, définie par

[Shoham 1987] pour qualifier une proposition qui dure sur deux intervalles consécutifs et qui dure sur leur réunion. L'oubli "actif" peut être rapproché du mécanisme *Erase* de PLANNER [Hewitt

1971] pour l'élimination des hypothèses fausses ou des assertions qui ne sont plus valides.

5.1.2.5. Une architecture opérationnelle pour une réutilisation limitée

Les deux systèmes proposent une architecture opérationnelle permettant la réutilisation effective dans un contexte restreint. Ils utilisent pleinement les trois mécanismes essentiels : l'héritage de bases de règles, le typage naturel et héritage de classe.

5.2. Points de divergence entre les deux systèmes

5.2.1. Sur la nature du problème traité

La principale différence est la nature du temps modélisé. Pour le système NéoGanesh, le temps est infini et réel. Pour le système d'analyse, le temps est irréel, et en boucle fermée. Ce dernier point est d'ailleurs essentiel pour la bonne marche du système dans un certain nombre de situations harmoniques, comme celles ou seule la fin du morceau permet d'assurer la stabilité harmonique du début (résolutions finales en particulier).

5.2.2. Sur les modèles et leurs représentations en NéOpus

En ce qui concerne la représentation en NéOpus, deux différences importantes sont à constater:

5.2.2.1. Deux représentations de l'oubli

Dans les deux cas l'oubli d'objets temporels a pour effet de rétrécir la mémoire à court terme et de considérer que l'histoire de certains objets n'est pas importante pour les étapes futures du raisonnement. Ceci est critiquable pour certaines applications, mais crucial pour des systèmes qui doivent être *très réactifs* dans leur raisonnement. Si les deux systèmes sont basés sur une représentation explicite de l'oubli d'information, celui-ci n'est pas représenté de la même manière. Dans NéoGanesh, l'oubli (passif) est représenté par des méta-règles, en utilisant l'architecture déclarative de contrôle. Dans le système d'analyse, l'oubli est représenté par des règles du domaine, exécutées en alternance avec les règles de reconnaissance.

5.2.2.2. Utilisation de l'héritage de bases de règles

Le système NéoGanesh repose principalement sur l'utilisation de l'héritage de bases de règles, et pas le système d'analyse. Ceci est du au fait que, pour le système NéoGanesh, une théorie générale de la perception des changements a été mise en évidence, mais pas pour le système d'analyse.

6. Conclusion: vers des patterns ou un méta-modèle ?

Nous avons volontairement structuré ce papier en proposant une généralisation *après* la description de deux systèmes, même si une structuration inverse aurait été plus pédagogique. Nous avons tenu à garder ce schéma de présentation pour deux raisons. D'abord la généralisation que nous proposons (un modèle général de représentation temporelle hiérarchique) n'est pas encore à un stade ou l'on pourrait parler de "framework". La réécriture des deux systèmes est cependant en cours (l'un dans le cadre d'un projet européen, l'autre dans le cadre d'une collaboration avec l'IRCAM) à la lumière de cette première analyse, et les résultats seront certainement riches d'enseignements. D'autre part il nous semble que la synthèse de ces deux expériences ne doit pas forcément conduire à l'élaboration d'une architecture qui permettrait de modéliser et construire toutes sortes de systèmes incluant une composante temporelle. Les facultés de réutilisation des langages de PPO (et en l'occurrence de NéOpus) sont limitées, et réutiliser un modèle temporel est une entreprise très ambitieuse. En revanche, il semble que deux types de synthèse sont particulièrement adaptés:

Vers des patterns pour des frameworks d'EOOPS

Dans le contexte de la PPO un consensus s'est récemment établi concernant les voies de recherche pour l'établissement de méthodologie de conception, basée sur une trilogie: 1) mécanismes (héritage, instanciation) 2) patterns de conception [Gamma et al.

1994] , et 3) frameworks

[Johnson and Foote

1988] ,

[Deutsch

1989],

[Pope

1989]. Si l'activité de recherche pour la fabrication de langages intégrant règles et objets est intense, il est visiblement encore trop tôt pour pouvoir identifier des patterns récurrents et spécifiques à ce type de représentation. Il est de même bien trop tôt pour proposer des méthodologies adaptées à ce style de représentation, même si le besoin est grand. Dans le domaine des systèmes de règles + objets, l'étude de réalisations substantielles exploitant les trois mécanismes clés des proposés ici devrait pouvoir, de manière similaire, dégager des "patterns de règles et objets", ou schémas de conception typiques pour une classe de sous problèmes donnés, afin de produire des "frameworks pour systèmes à règles et objets". Il semble bien que le schéma agrégation/oubli fournisse une première ébauche dans ce sens, qui demande encore à être creusée.

Vers un méta-modèle objet des ontologies temporelles

Le deuxième effort de synthèse de ces deux systèmes est l'élaboration d'un *méta-modèle* commun, en utilisant la technologie METAGEN [Revault et al.

1995]. Ce méta-modèle devra permettre de spécifier les caractéristiques des deux ontologies temporelles et leur interprétations. Cette méta-modélisation, si elle réussit, devrait permettre d'abstraire ces expériences en produisant un composant logiciel effectivement réutilisable qui pourrait alors grandement faciliter la conception et la fabrication de systèmes de raisonnement temporel complexes.

7. Bibliographie

[Albert 1994] **Albert P**. *Il*

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

[Ash and Hayes-Roth 1990] **Ash D and Hayes-Roth B.** Temporal representation in backboard architectures, Technical Report Stanford University, Ca., KSL 90-16, , 1990.

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

[Baroni 1984] **Baroni M C, L.** Musical Grammars and Computer Analysis. Florence, Italy, Olschki 1984.

[Barry and McAffer 1992] **Barry B and McAffer J**. *ENVY/Expert : a rule-based programming system for Smalltalk*. Canadian AI Conference, 1992.
[Beaudoin 1990] **Beaudoin P**. Jazz, mode d'emploi. Paris, Outre Mesure 1990.
[Bitsch 1957] **Bitsch M**. Précis d'harmonie tonale. Paris, Editions musicales Alphonse Leduc 1957.
[Carré et al. 1995] **Carré B, et al.** *Classification et Objets: programmation ou représentation ?* Journées nationales du PRC IA, Nancy, p. 213-237,1995.
[CLIPS 1994] **CLIPS**. Third Conference on CLIPS. Lyndon B. Johnson Space Center, NASA 1994.
[Coker 1964] **Coker J**. Improvising Jazz. Englewood Cliffs, New Jersey, Prentice-Hall 1964.
[Crawford et al. 1994] **Crawford J M, et al.** *Embedded Object-Oriented Production Systems*. OOPSLA Workshop on EOOPS, Portland, Laforia-IBP, p. 57-62,1994.
[Crawford and Kuipers 1991] **Crawford J M and Kuipers B J**. *Negation and proof by contradiction in Access-Limited Logic*. Ninth Conference on Artificial Intelligence, 1991.
[Dechter et al. 1991] **Dechter R, Meiri I, Judea P**. Temporal constraints networks. Artificial Intelligence 4 : 61-95, 1991.
[Deliège 1994] **Deliège**. *Session on "Lerdalh & Jackendoff : 10 years on*. International Conference on Music Perception and Cognition (ICMPC), Lièges (Belgium), 1994.
[Deutsch 1989] **Deutsch L P**. Design Reuse and Frameworks in the Smalltalk-80 Programming System. Software Reusability 2 : 55-71, 1989.
[Dojat et al. 1992] **Dojat M, Brochard L, Lemaire F, Harf A**. A knowledge-based system for assisted ventilation of patients in intensive care. Int J Clin Monit Comput 9 : 239-250, 1992.
[Dojat et al. 1995] **Dojat M, et al.** Clinical evaluation of a knowledge-based system providing ventilatory management and decision for extubation during weaning from mechanical ventilation. Am. J. Respir. Crit. Care Med. : to appear, 1995.
[Dojat and Pachet 1992]**Dojat M and Pachet F**. *An extendable knowledge-based system for the control of mechanical ventilation*. 14th IEEE-EMBS, Paris, p. 920-921,1992.
[Dojat and Pachet 1995]**Dojat M and Pachet F**. *Three Compatible Mechanisms for Representing Medical Context Implicitly*. IJCAI Workshop on "Context in Knowledge and Reasoning Modelling", Montréal, p. 69-78,1995.
[Dojat and Sayettat 1995] **Dojat M and Sayettat C**. A realistic model for temporal reasoning in real-time patient monitoring. AAI : to appear, 1995.
[Dvorak 1994]**Dvorak J**. *Two perspectives on the EOOPS theme*. OOPSLA Workshop on EOOPS, Portland, Laforia-IBP, p. 49-55,1994.
[Fake 1983] **Fake**. The World's Greatest Fake book. San Franscisco, Sher Music Co 1983.
[Forgy 1994] **Forgy C**. *Embedded Object-Oriented Production Systems*. OOPSLA Workshop on EOOPS, Portland, Laforia-IBP, p. 11-16,1994.
[Forgy 1982] **Forgy C L**. Rete: A fast algorithm for the many pattern/ many object pattern match problem. AI 19 : 17-37, 1982.
[Gamma et al. 1994] **Gamma E, Helm R, Johnson R, Vlissides J**. Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley 1994.
[Hewitt 1971] **Hewitt C.** Description and theoritical analysis (using schemata) of PLANNER: a language for proving theorems and manipulating models in robots, Technical Report MIT, AI-TR-258, , 1971.

[Johnson and Foote 1988] **Johnson R E and Foote B**. Designing Reusable Classes. Journal of Object-Oriented Programming 1 : 22-25, 1988.
[Johnson-Laird 1991] **Johnson-Laird P N**. Jazz Improvisation: A Theory at the Computational level. Representing Musical Structures, Academic Press**:** 291-325, 1991.
[Kohane 1987] **Kohane I.** Temporal reasoning in medical expert systems. Phd Thesis, MIT, Computer Science, 1987.
[Kowalski and Sergot 1986] **Kowalski R A and Sergot M J**. A logic-based calculus of events. 4 : 67-95, 1986.
[Laske 1993] **Laske O**. In search of a Generative Grammar for Music. Machine models of Music. S. M. Schwanauer and D. A. Levitt, MIT Press**:** 215-240, 1993.
[Lau and Vincent 1993] **Lau F and Vincent D D**. Formalized decision support for cardiovascular intensive care. Comput. Biomed. Res. 26 : 294-309, 1993.
[Lerdahl and Jackendoff 1983] **Lerdahl F and Jackendoff R**. A Generative Theory of Tonal Music. Cambridge, MIT Press 1983.
[Maxwell 1992] **Maxwell H J**. An Expert System for Harmonizing Analysis of Tonal Music. Understanding Music with AI: Perspectives on Music Cognition. K. E. O. L. M. Balaban, AAAI Press**:** 335-353, 1992.
[McDermott 1982] **McDermott D**. A temporal logic for reasoning about processes and plans. Cognitive science 6 : 101-155, 1982.
[Miller 1956] **Miller G A**. The magical number seven, plus or minus two: some limits on our capacity for processing information. Psychol Rev 63 : 81-97, 1956.
[Miranker 1994] **Miranker D**. *Encapsulating Rules*. OOPSLA Workshop on EOOPS, Portland, Laforia-IBP, p. 23-26,1994.
[Missaoui and Gul 1988] **Missaoui R and Gul T.** Is Object Oriented Programming Well Suited to Frame Implementation, Technical Report Université du Québec à Montréal, Rapport de recherche 71, 1988, 1988.
[Murray and Balaban 1988] **Murray N V and Balaban M.** The logic of time structures: temporal and nonmonotonic features, Technical Report University of New Jersey at Albany, 88-32, , 1988.
[Napoli and Volle 1993] **Napoli A and Volle P.** Une introduction aux logiques terminologiques, Technical Report CRIN, CRIN 93-R-033, 1993, 1993.
[Nebel 1990] **Nebel B**. Reasoning and Revision in Hybrid Representation Systems., Springer-Verlag 1990.
[Pachet 1992a] **Pachet F.** Représentation de connaissances par objets et règles : le système NéOpus. Phd Thesis, Université Paris 6, Ph. D., 1992a.
[Pachet 1992b] **Pachet F**. *Rule Base Inheritance*. Colloque "Représentations Par Objets", La grande Motte, p. 187-200,1992b.
[Pachet 1994a] **Pachet F**. *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, p. 195-201,1994a.
[Pachet 1994b] **Pachet F**. *An object-oriented representation of pitch-classes, intervals, scales and chords*. Premières Journées d'Informatique Musicale, LaBRi, Université de Bordeaux, p. 19-34,1994b.
[Pachet 1994c] **Pachet F.** Proceedings of the OOPSLA'94 Workshop on EOOPS, Technical Report Laforia-IBP, 94/24, novembre 1994, 1994c.
[Pachet 1994d] **Pachet F**. *Vers un modèle du raisonnement dans les langages à objets*. Colloque "Langages et Modèles à Objets", Grenoble, p. 111-123,1994d.
[Pachet and Perrot 1994] **Pachet F and Perrot J-F**. *Rule Firing with Metarules*. Software Engineering and Knowledge Engineering - SEKE '94, Jurmala, Latvia, p. 322-329,1994.

[Pachet and Roy 1995] **Pachet F and Roy P**. *Mixing constraints and objects: a case study in automatic harmonization*. TOOLS Europe '95, Versailles, Prentice-Hall, p. 119-126,1995.
[Paquette et al. 1995] **Paquette G, Pachet F, Giroux S**. EpiTalk, a generic tool for the development of advisor systems (to be published). Artificial Intelligence in Education , 1995.
[Patel-Schneider 1990] **Patel-Schneider P F**. Practical, Object-Based Knowledge Representation for Knowledge-Based Systems. Information Systems 15 (1): 9-19, 1990.
[Pope 1989] **Pope S**. Machine Tongues XI: Object-Oriented Software Design. Computer Music Journal 13 : 9-22, 1989.
[Ramalho and Ganascia 1994] **Ramalho G and Ganascia J-G**. *Simulating Creativity in Jazz Performance*. 12th AAAI Conference, Seattle, 1994.
[Ramalho and Pachet 1994] **Ramalho G and Pachet F**. *From Real Book to Real Jazz Performance*. International Conference on Music Perception and Cognition (ICMPC), Lièges (Belgium), p. 349-350,1994.
[Rathke 1991] **Rathke C**. *Implementing Frames in an Object-Oriented Programming Language*. East EurOOpe, Bratislava, p. 88-94,1991.
[Real 1981] **Real**. The Real Book, The Real Book Press 1981.
[Revault et al. 1995] **Revault N, Sahraoui H A, Blain G, Perrot J-F**. *A Metamodeling technique: the MétaGen system*. TOOLS Europe '95, Versailles, Prentice-Hall, p. 127-139,1995.
[Roads 1988] **Roads C**. Grammars as Representations for Music. Foundations of Computer Music. C. Roads and J. Strawn, MIT Press: 403-442, 1988.
[Roche 1989] **Roche C**. *From Objects to Frames: Artificial Intelligence Knowledge Representation in Smalltalk-80*. TOOLS, Paris, SOL, p. 177-185,1989.
[Rolland and Pachet 1995] **Rolland P-Y and Pachet F**. *Représentation de connaissances et programmation de synthétiseurs*. Deuxièmes Journées d'Informatique Musicale, Paris, p. 123-132,1995.
[Rucker et al. 1990] **Rucker D W, Maron D J, Shortliffe E H**. Temporal representation of clinical algorithms using expert system and database tools. Computers and biomedical research 23 : 222-239, 1990.
[Sayettat 1990] **Sayettat C.** Contributions à la représentation du changement. Phd Thesis, Université de Nantes, Habilitation à Diriger les Recherches, 1990.
[Schank 1991] **Schank R C**. Where's the AI. AI Magazine (Winter): 38-49, 1991.
[Shahar 1994] **Shahar Y.** A Knowledge-Based Method for Temporal Abstraction of Clinical Data. Phd Thesis, Stanford, Ph.D., 1994.
[Shahar and Musen 1993] **Shahar Y and Musen M A**. RESUME: A temporal-abstaction system for patient monitoring. Comput Biomed Res 26 : 255-273, 1993.
[Shoham 1987] **Shoham Y**. Temporal logic in AI: semantical and onthological considerations. Artificial Intelligence 33 : 89-104, 1987.
[Shortliffe 1990] **Shortliffe E H**. Clinical decision support systems in medical informatics. Computer applications in health care. E.H.Shortliffe and A. Perrault, Addison Wesley: 466-501, 1990.
[Steedman 1984] **Steedman M J**. A Generative Grammar for Jazz Chord Sequences. Music Perception 2 (1): 52-77, 1984.
[Steels 1979] **Steels L.** Reasoning modeled as a Society of Communicating Experts, Technical Report MIT AI Lab., AI-TR-542, , 1979.
[Tsang 1987] **Tsang E P**. *Times structures for Artificial Intelligence*. 10th IJCAI, p. 456-461,1987.

[Ulrich 1977] **Ulrich W**. *The Analysis and Synthesis of Jazz by Computer*. Fifth International Joint Conference on Artificial Intelligence, MIT, Cambridge, Ma, p. 865-872,1977.
[Van Beek 1992] **Van Beek P**. Reasoning about qualitative temporal information. A.I. 58 : 297-326, 1992.
[Vilain and Kautz 1986]**Vilain M and Kautz H**. *Constraint propagation algorithms for temporal reasoning.* AAAI, Philadelphia, p. 377-382,1986.
[Williamson and Hanks 1993] **Williamson M and Hanks S**. *Exploiting domain structure to achieve efficient temporal reasoning*. 13th IJCAI, Chambéry (Fr), p. 152-157,1993.
[Winograd 1993] **Winograd T**. Linguistic and Computer Analysis of Tonal Harmony. Machines Models of Music (reprint). S. M. Schwanauer. and D. A. Levitt, MIT Press**:** 113-153, 1993.
[Yelland 1992] **Yelland P**. *Experimental Classification Facilities for Smalltalk.* OOPSLA 92, Vancouver, ACM Sigplan, p. 235-246,1992.